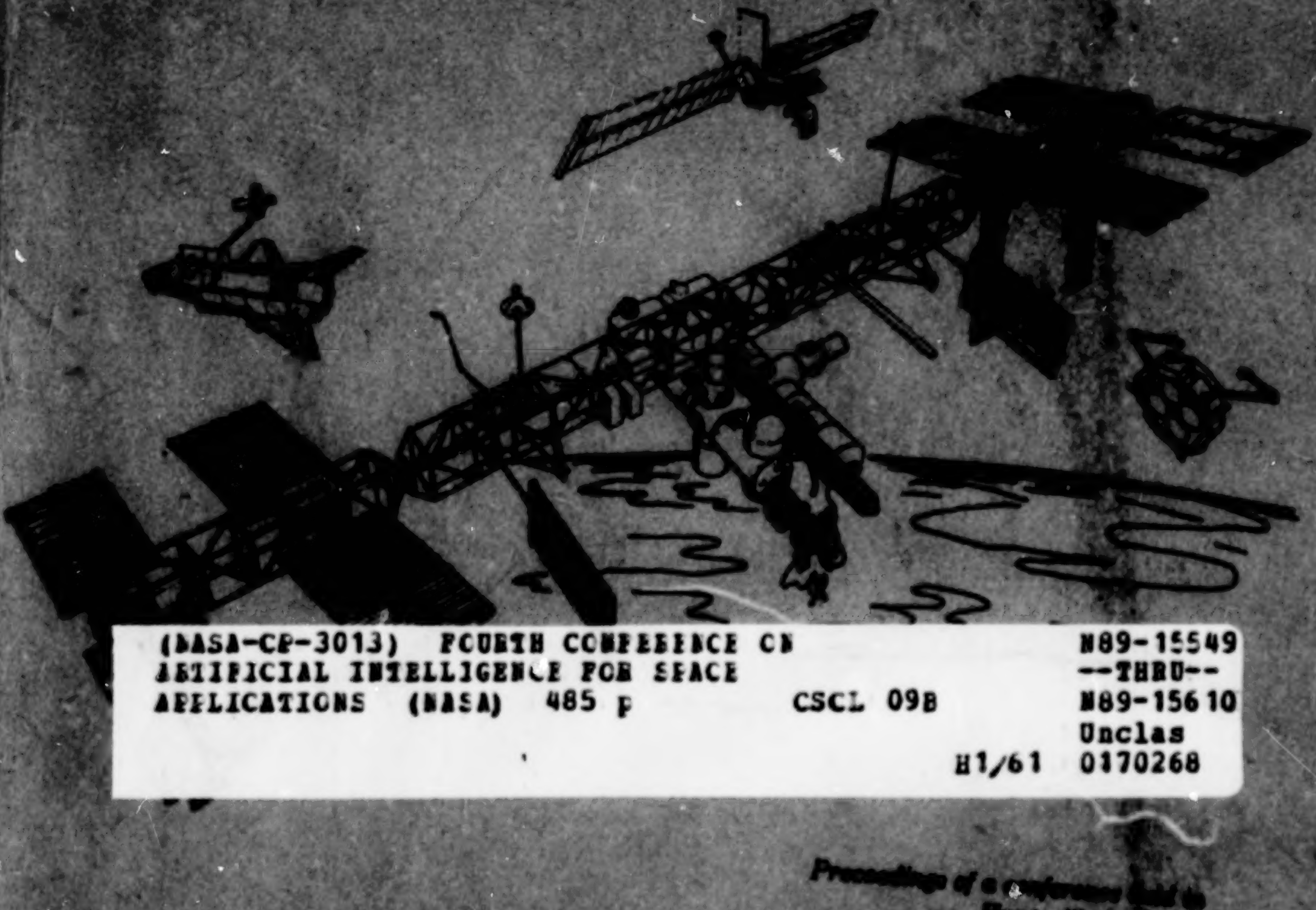
[illegible]

Fourth Conference on Artificial Intelligence for Space Applications



```
(NASA-CP-3013)  FOURTH CONFERENCE ON                      N89-15549
ARTIFICIAL INTELLIGENCE FOR SPACE                         --THRU--
APPLICATIONS  (NASA)  485 p          CSCL 09B             N89-15610
                                                          Unclas
                                              H1/61       0170268
```

Proceedings of a conference held in
Huntsville, Alabama
November 15 and 16, 1988

Sponsored by:
The University
Of Alabama
in Huntsville
[illegible]

George C. Marshall Space Flight Center
Marshall Space Flight Center, Alabama [illegible]

NASA Conference Publication 3013

Fourth Conference on Artificial Intelligence for Space Applications

Compiled by
S. L. O'Dell
J. S. Denton
and M. Vereen
George C. Marshall Space Flight Center
Marshall Space Flight Center, Alabama

Proceedings of a conference sponsored by
the University of Alabama in Huntsville and
the National Aeronautics and Space Administration
and held in Huntsville, Alabama
November 15 and 16, 1988

NASA
National Aeronautics
and Space Administration

Scientific and Technical
Information Branch

1988

Foreword

During the past four years, interest in the use of Artificial Intelligence technology to support America's exploration of space has continued to grow. Much of this increased interest can be attributed to the successful development of prototype knowledge-based systems built by the National Aeronautics and Space Administration (NASA), NASA contractors, and other members of the aerospace community. Not only do these prototype systems demonstrate the feasibility of applying AI technology to the solution of specific problems but also they serve to expose technical decision makers to the potential of this technology in other contexts. This effect is noted in the final report of the Space Station Advanced Automation Study, a study which was conducted by a small group of the country's most experienced practitioners of knowledge-based systems and which was conducted for the Space Station Level I Strategic Plans and Programs Division, Office of Space Station, NASA Headquarters. Indeed, this "blue ribbon" group, formed to analyze the current and future potential of knowledge-based systems on Space Station, found the climate for the acceptance of this technology to be "very good" and the attitude of future Space Station crew members towards these systems to be "overwhelmingly positive."

NASA's Marshall Space Flight Center (MSFC) and the University of Alabama in Huntsville (UAH) are sponsoring the Fourth Conference on Artificial Intelligence for Space Applications, AISA'88, to promote discussion of the possibilities created by this work. Additionally, the sponsors seek to provide an opportunity for those who apply artificial intelligence methods to space related problems to identify common goals, to compare the effectiveness of the various approaches being employed, and to discuss issues of general interest. Towards these ends, the technical/program committee chose sixty papers to be presented in the eighteen technical sessions of the conference. These eighteen sessions cover a broad range of topics in the knowledge-based systems area and also consider hardware issues, robotics applications, and vision concerns to name a few other topics.

The *Proceedings of the Fourth Conference on Artificial Intelligence for Space Applications* contain the papers (or abstracts of papers) presented at AISA'88. The sponsors hope that these proceedings will be useful as a reference to various AI activities in progress and will contribute to the literature of applications of Artificial Intelligence.

Judith S. Denton

PRECEDING PAGE BLANK NOT FILMED

Fourth Conference on Artificial Intelligence for Space Applications

Huntsville, Alabama
1988 November 15-16

Conference Director

Judith Denton, NASA/MSFC

Technical/Program Co-Chairpersons

Michael Freeman, NASA/MSFC
James Johannes, UAH

Technical Committee

Rowland Burns, NASA/MSFC
Elaine Hinman, NASA/MSFC
Bill Selig, NASA/MSFC
Dan Hays, UAH
Caroline Wang, NASA/MSFC
Greg Franks, NASA/MSFC
Bryan Walls, NASA/MSFC
Bernard Schroer, UAH
Pat Ryan, UAH

Coordinators

AI Working Group
Mary Vereen, NASA/MSFC

Brochures and Programs
Diane Vaughan, NASA/MSFC

Logistics
James Parker, NASA/MSFC

NASA/MSFC Attendance
David Allen, NASA/MSFC

NASA Participation
Thomas Dollman, NASA/MSFC

Professional Exhibits
Michael Whitley, NASA/MSFC

Proceedings
Stephen O'Dell, NASA/MSFC

Publicity
Frank Vinz, NASA/MSFC

Tutorials
Gerry Higgins, NASA/MSFC

Vendors
Gary Workman, UAH

Arrangements

Karen Mack, UAH
Kathy Landman, UAH

Advisor

Gabriel Wallace, NASA/MSFC

PRECEDING PAGE BLANK NOT FILMED

TABLE OF CONTENTS

REAL TIME

Considerations in Development of Expert Systems for Real-Time Space Applications [ABSTRACT]
S. Murugesan 1

Real-Time Control for Manufacturing Space Shuttle Main Engines: Work in Progress
Corinne C. Ruokangus 5

Solutions to Time Variant Problems of Real-Time Expert Systems
Show-Way Yeh, Chuan-lin Wu, and Chaw-Kwei Hung 19

DESIGN KNOWLEDGE CAPTURE

Functional Reasoning in Diagnostic Problem Solving
Jon Sticklen, W. E. Bond, and D. C. St. Clair 29

The Elements of Design Knowledge Capture
Michael S. Freeman 39

KAM: A Tool to Simplify the Knowledge Acquisition Process
Gary A. Gettig 47

MANAGEMENT

Design of an Expert System for Estimating the Cost of New Knowledge in High-Energy Astrophysics [ABSTRACT]
Edward L. Bosworth, Jr., and A. J. Fennelly 57

Issues in Management of Artificial Intelligence Based Projects
P. A. Kiss and Michael S. Freeman 59

Development of an Expert Planning System for OSSA [ABSTRACT]
B. Groundwater, M. F. Lembeck, and L. Sarsfield 67

ROBOTICS

Planning Actions in Robot Automated Operations
A. Das 69

Integration of Task Level Planning and Diagnosis for an Intelligent Robot
Arthur Gerstenfeld 75

A Graphical, Rule Based Robotic Interface System
James W. McKee and John Wolfsberger 85

PRECEDING PAGE BLANK NOT FILMED

KNOWLEDGE REPRESENTATION

PDA: A Coupling of Knowledge and Memory for Case-Based Reasoning [ABSTRACT]
S. Bharwani, J. Walls, and E. Blevins 93

Approximate Spatial Reasoning [ABSTRACT]
Soumitra Dutta 95

Representation and Matching of Knowledge to Design Digital Systems
J. U. Jones and S. G. Shiva 97

FAULT DIAGNOSTICS I

Vulnerability-Attention Analysis for Space-Related Activities
Dan Hays, Sung Yong Lee, and John Wolfsberger 107

Graph-Based Real-Time Fault Diagnostics
S. Padalkar, G. Karsai, and J. Sztipanovits 115

Automatic Detection of Electric Power Troubles (ADEPT)
Caroline Wang, Hugh Zeanah, Audie Anderson, Clint Patrick, Mike Brady, and Donnie Ford 125

AUTOMATIC PROGRAMMING

An Overview of Very High Level Software Design Methods
Maryam Asdjodi and James W. Hooper 131

Artificial Intelligence Approaches to Software Engineering
James D. Johannes and James R. MacDonald 141

Automatic Programming for Critical Application [ABSTRACT]
Raj L. Loganantharaj 151

Using Automatic Programming for Simulating Reliability Network Models
Fan T. Tseng, Bernard J. Schroer, S. X. Zhang, and John W. Wolfsberger 153

KNOWLEDGE BASE / DATA BASE

Object Oriented Studies into Artificial Space Debris
J. M. Adamson and G. Marshall 163

Extending the Data Dictionary for Data/Knowledge Management
Cecile L. Hydrick and Sara J. Graves 173

Case-Based Reasoning: The Marriage of Knowledge Base and Data Base
Kirt Pulaski and Cyprian Casadaban 183

Expert System Validation in Prolog [ABSTRACT]
Todd Stock, Rolf Stachowitz, Chin-Liang Chang, and Jacqueline Combs 191

SCHEDULING

Expert System for On-Board Satellite Scheduling and Control
John M. Barry and Charisse Sary 193

Dypas: A Dynamic Payload Scheduler for Shuttle Missions [ABSTRACT]
Stephen Davis 205

A Knowledge-Based Decision Support System for Payload Scheduling
Rajesh Tyagi and Fan T. Tseng 207

A CLIPS Prototype for Autonomous Power System Control
James M. Vezina and Leon Sterling 211

VISION

A Hardware Implementation of a Relaxation Algorithm to Segment Images
Antonio G. Loda' and Heggere S. Ranganath 221

Using AGNESS (A Generalized Network-based Expert System Shell) for Matching Images [ABSTRACT]
Ting-Chuen Pong, Chung-Mong Lee, and James Slagle 231

Automatic Inspection of Analog and Digital Meters in a Robot Vision System
Mohan M. Trivedi, Suresh Marapane, and Chu Xin Chen 233

DESIGN KNOWLEDGE CAPTURE II

Knowledge-Based Approach to System Integration [ABSTRACT]
W. Blokland, C. Krishnamurthy, C. Biegl, and J. Sztipanovits 243

Successful Expert Systems for Space Shuttle Payload Integration
Keith Morris 245

Automated Knowledge Base Development from CAD/CAE Databases
R. Glenn Wright and Mary Blanchard 253

LEARNING

Dynamic Reasoning in a Knowledge-Based System
Anand S. Rao and Norman Y. Foo 261

Strategies for Adding Adaptive Learning Mechanisms to Rule-Based Diagnostic Expert Systems
D. C. St. Clair, C. L. Sabharwal, W. E. Bond, and Keith Hacke 271

An Architecture for an Autonomous Learning Robot
Brian Tillotson 281

ROBOTICS DESIGN USING SIMULATION

Toward a Computational Theory for Motion Understanding: The Expert Animators Model
Ahmed S. Mohamed and William W. Armstrong 289

Graphic Simulation Test Bed for Robotics Applications in a Workstation Environment
J. Springfield, A. Mutammara, G. Karsai, G. E. Cook, J. Sztipanovits, and K. Fernandez 303

Design of a Simulation Environment for Laboratory Management by Robot Organizations
Bernard P. Zeigler, Francois E. Cellier, and Jerzy W. Rozenblit 313

EXPLANATION SYSTEMS

Explanation Production by Expert Planners
Susan Bridges and James D. Johannes 323

Knowledge Representation Issues for Explaining Plans
Mary Ellen Prince and James D. Johannes 331

Simple Explanations and Reasoning: From Philosophy of Science to Expert Systems
Daniel Rochowiak 341

TELEMETRY MONITORING

Controlling Basins of Attraction in a Neural Network-Based Telemetry Monitor
Benjamin Bell and James L. Eilbert 349

An Expert System for Satellite and Instrument Data Anomaly and Fault Isolation
Carl Busse 356

A Multiprocessing Architecture for Real-Time Monitoring [ABSTRACT]
James L. Schmidt, Simon M. Kao, Jackson Y. Read, Scott M. Weitzenkamp and Thomas J. Laffey 369

FUTURE SPACE APPLICATIONS

PI-in-a-Box: Intelligent Onboard Assistant for Spaceborne Experiments in Vestibular Physiology
Silvano Colombano, Laurence Young, Nancy Wogrin, and Don Rosenthal 371

Artificial Intelligence Applications in Space and SDI - A Survey
Harvey E. Fiala 381

Concepts for Autonomous Flight Control for a Balloon on Mars
Thomas F. Heinsheimer, Robyn C. Friend, and Neil G. Siegel 391

A Very Large Area Network (VLAN) Knowledge-Base Applied to Space Communication Problems
Carol S. Zander 401

Ada in AI or AI in Ada? On Developing a Rationale for Integration
Philippe E. Collard and Andre Goforth 411

PLANNING

Automated Scheduling and Planning (ASAP) in Future Ground Control Systems
Sam Matlin 421

A Dynamic Case-Based Planning System for Space Station Application
F. Oppacher and D. Deugo 431

Towards a Knowledge-Based System to Assist the Brazilian Data-Collecting System Operation
V. Rodrigues, P. O. Simoni, P. P. B. Oliveira, C. A. Oliveira,
and C. A. M. Nogueira 441

FAULT DIAGNOSTICS II

Use of an Expert System Data Analysis Manager for Space Shuttle Main Engine Test Evaluation
Ken Abernethy 451

Spacecraft Environmental Anomalies Expert System
H. C. Koons and D. J. Gorney 457

Using Hypermedia to Develop an Intelligent Tutorial/Diagnostic System for the Space Shuttle Main Engine Controller Lab
Daniel O'Reilly, Robert Williams, and Kevin Yarbrough 467

Object-Oriented Fault Tree Evaluation Program for Quantitative Analysis
F. A. Patterson-Hine and B. V. Koen 477

ADDENDA

Considerations in Development of Expert Systems for Real-Time Space Applications
S. Murugesan 487

INDEX OF AUTHORS 497

N89-15550

Considerations in Development of Expert Systems for Real-Time Space Applications*

S. Murugesan
NASA Ames Research Center
Moffett Field, CA 94035

Abstract

Over the years demand on space systems have been increased tremendously and this trend will continue for the near future. The enhanced capabilities of space systems, however, can only be met with increased complexity and sophistication of onboard and ground systems, and artificial intelligence and expert system concepts have a significant role in space applications.

Expert systems could facilitate autonomous decision making, improved fault diagnosis and repair, enhanced performance and less reliance on ground support. However, some requirements have to be fulfilled before practical use of flight-worthy expert systems for onboard (and ground) operations.

This paper discusses some of the characteristics and important considerations in design, development, implementation and use of expert systems for real-life space applications. Further, it describes a typical life cycle of expert system development and its usage.

Characteristics: Expert systems for real-time critical space applications need to have the following characteristics:

- Robustness
- Real-time execution (high execution speed compatible with physical requirements)
- Real-world inputs from various subsystems in operation
- Interactive inputs from operators, other expert systems and ground commands
- Extremely high reliability of operation in their application environment
- High degree of correctness and consistency of decisions

** The paper by S. Murugesan appears in this volume, beginning on page 487.*

- Testability under various operational modes, failures and credible contingencies
    - Validation and Verification (V&V) and "proof-of-concept"
- Tolerance to failures of hardware, software (knowledge base, inference engine, operating system, etc.), input/output networks and monitoring devices
- Coordination with other expert systems; interaction with a common data/knowledge base
- Fail-safe operations and graceful performance degradation
- Realisability with minimum hardware and power consumption meeting other MIL-STD requirements
- Symbolic and data processing capabilities
- Accomadability to change/modification

Development: Development of expert system should be considered as a system engineering activity encompassing many tasks. The life cycle model of expert system is somewhat different from well-accepted software life cycle, though there are many commonalities. Major phases in life cycle of expert system development include:

- Problem identification/specification
- Acquisition of domain knowledge from experts, previous case history, operation and design documents
- Formulation of knowledge base, knowledge representation
- Choice (and/or development) of suitable inferencing/reasoning schemes and procedures
- Testing of expert system software (residing in development tools) under static (nonreal-time) and real-time environments
    - Review human domain experts and specialists; revisions

- Integration of hardware deliverables and compiled 'expert software'
- Testing under simulated and real-life environments under various operational and failure modes
  - Reviews by human domain experts and specialists; revisions
- Delivery of flight-worthy Expert System; maintenance and upgradation

---

* *The paper by S. Murugesan appears in this volume, beginning on page 487.*

N89 - 15551

Real-Time Control for Manufacturing Space Shuttle Main Engines:

Work In Progress

Corinne C. Ruokangas
Rockwell Science Center, Palo Alto Laboratory
444 High Street, Suite 400
Palo Alto, CA 94301
ruokangas@score.stanford.edu

ABSTRACT

During the manufacture of space-based assemblies such as Space Shuttle Main Engines, flexibility is required due to the high-cost and low-volume nature of the end products. Various systems have been developed pursuing the goal of adaptive, flexible manufacturing for several space applications, including an Advanced Robotic Welding System [Sliwinski 87] for the manufacture of complex components of the Space Shuttle Main Engines. The Advanced Robotic Welding System (AROWS) is an on-going joint effort, funded by NASA, between NASA/Marshall Space Flight Center, and two divisions of Rockwell International: Rocketdyne and the Science Center. AROWS includes two levels of flexible control of both motion and process parameters: Off-line programming using both geometric and weld-process data bases, and real-time control incorporating multiple sensors during weld execution. Both control systems were implemented using conventional hardware and software architectures. The feasibility of enhancing the real-time control system using the problem-solving architecture of Schemer [Ruokangas 88] is being investigated and is described in this paper.

Schemer is a knowledge-based system designed to provide more complex real-time control by supporting real-time response to multiple and conflicting interrupts, and problem solving under time and resource constraints; on-going research supports the incorporation of techniques from decision analysis. The Schemer architecture is event driven and is similar to a blackboard system. It supports the interruption, suspension, and resumption of problem solving tasks, concepts which are essential to providing real-time response to changes in the environment [Fehling 87]. Schemer provides support for intelligent real-time modification of Off-line programming plans and resolution of sensor conflicts.

This paper summarizes the development of a prototype system for simulation of real-time control of robot motion and the welding process using multiple sensors. The system provides simulation of prioritized, event-driven responses to multiple sensors affecting multiple processes, with the sensors providing both cooperative and conflicting information. A single vision sensor is used to modify robot motion. In parallel with modifications of motion, multiple penetration sensor systems are simulated to affect weld-current values. Further development of the system will include additional sensor fusion techniques such as decision theory methods, and plan transformation rules. While the current implementation is on a Symbolics 3600 series system in Common Lisp, other hardware platforms are being evaluated for additional studies towards implementation in manufacturing.

PRECEDING PAGE BLANK NOT FILMED

INTRODUCTION

AI in Space Applications

There is widespread interest throughout the aerospace community in the application of Artificial Intelligence (AI) developments to both space and earth based activities, including efforts in planning, scheduling, and real-time control. Effective utilization of key AI components can support NASA in future space exploration, including Space Station and Mars Rover projects, satellites, and spacecraft communications and control.

"The primary goal of AI is to make machines smarter" [Winston 87], that is, to provide more autonomous systems which are more useful, competent and less demanding of human interaction. Significant development of space exploration and space-related projects, both flight and ground based, relies on the development and implementation of autonomous systems. An autonomous system, operating in a complex, dynamic environment such as space or manufacturing, should have the capability to both define and execute plans to achieve its goals. In many environments, limitations of time, information, and other critical resources constrain the determination and use of the plans. The system must be able to manage its reasoning and other activities to make the best use of available resources. Problem-solving under these conditions has been referred to as resource-bounded problem-solving - "controlling and adapting problem-solving actions to meet critical, contextually-determined constraints" [Fehling 88]. One arena in which resource-bounded problem-solving autonomous systems will prove invaluable is the manufacturing of space-based assemblies which must be robust enough to operate in remote locations. This paper discusses the development of such a system, to be applied to the manufacture of Space Shuttle Main Engine components.

A research area which holds promise for influencing advancements in autonomous systems is intelligent real-time control, or problem-solving under time constraints. Commercial AI shells, designed for the development of consulting type systems, provide inadequate support for real-time process control systems, i.e. those which require problem solving in a dynamic processing environment, in an interruptible and prioritized event-driven manner. Implementation of intelligent, flexible real-time control in real world environments has proven to be a considerable challenge to the AI research community. The resource-bounded problem-solving architecture described in this paper addresses several of the fundamental challenges intrinsic to intelligent real-time control, such as appropriate response to multiple and varying priorities of interrupts, and conflict resolution.

SSME Background

The flexible real-time control architecture described in this paper can be applied in various areas of motion and process control throughout manufacturing and aerospace activities. The specific application discussed here is the automated welding of complex parts of the Space Shuttle Main Engine (SSME), which could be advanced by a high degree of run-time adaptivity both for seam following and weld process modification. The SSME is a low-volume, high precision product. Because part geometries vary significantly, it is difficult to accurately predict the variations in geometric position and process parameters defined in a robotic welding schedule. During weld execution, the welding process itself can induce heat distortions, further complicating the task of automatic welding. In 1983, a development project was initiated by NASA Marshall Space Flight Center to support robot-based welding of components of the SSME. At that time, commercial robot systems did not provide either the desired adaptive real-time control nor off-line programming for motion and process control; the goal of the project was development of a technology base for penetration sensors, seam-tracking sensors, and integrated off-line generation of process commands [Sliwinski 87]. The Advanced RObotic Welding System (AROWS), developed by the cooperative effort of two Rockwell divisions and demonstrated for NASA in late 1987, incorporated two levels of adaptive control, as shown in figure 1.

- Off-line Programming (OLP) was based on existing geometric data bases and weld parameter data bases developed by interaction with Rocketdyne welding engineers. The overall function of the OLP system was initial generation of both motion and process commands, using graphical simulation.

- The adaptive real-time control system included the development and implementation of both motion correction and weld penetration sensors, as well as the Sensor Controller for overall coordination of the sensor subsystems and control of the robot and weld parameters during weld execution.

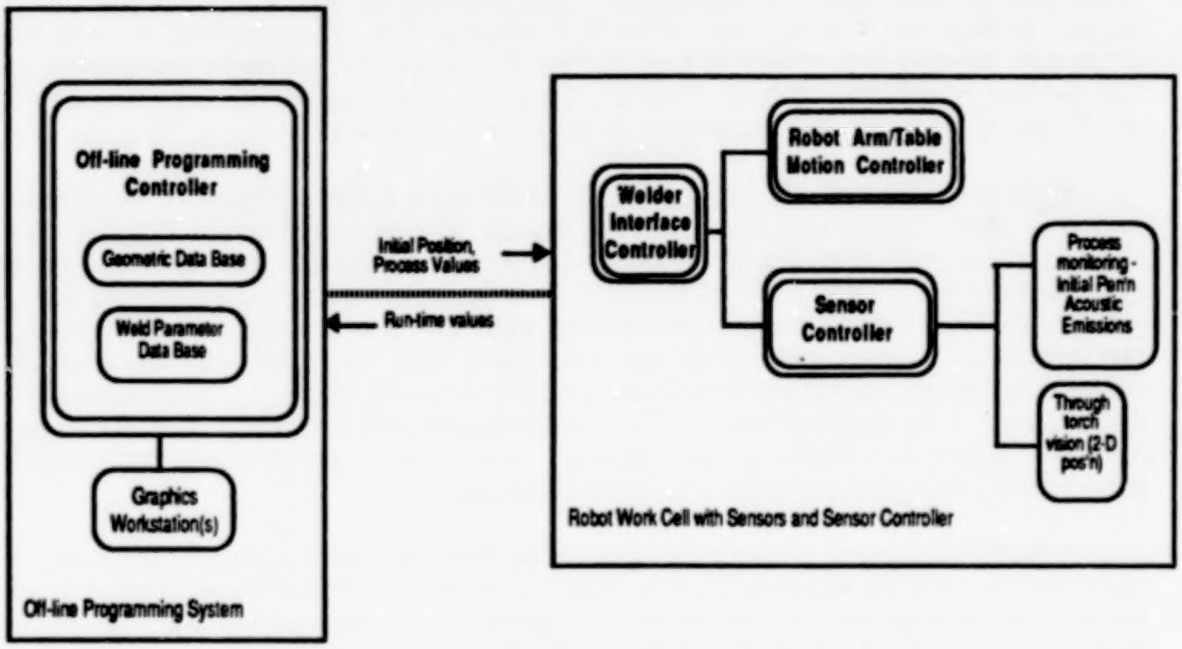

Off-line Programming Controller
Geometric Data Base
Weld Parameter Data Base
Graphics Workstation(s)
Off-line Programming System
Initial Position, Process Values
Run-time values
Welder Interface Controller
Robot Arm/Table Motion Controller
Sensor Controller
Process monitoring - Initial Pen'n Acoustic Emissions
Through torch vision (2-D pos'n)
Robot Work Cell with Sensors and Sensor Controller

Figure 1: The adaptive control elements of the Advanced Robotic Welding System: Off-Line Programming and Real-time control

AI for Control Applications: Schemer

Conventional control and programming environments are often limited in several key areas. In commercially available AI shells, techniques are used which surmount several of these limitations; however, these shells are primarily designed to develop consultative, off-line systems rather than on-line or real-time systems. Schemer supports the application of AI techniques to real-time problem-solving environments. Specifically, Schemer is being used to enhance the real-time control of a robot and welding equipment for manufacture of the Space Shuttle Main Engines, utilizing a simulation environment developed at Rockwell Palo Alto Laboratory.

Limitations to Conventional Systems: Conventional control theory is frequently limited in its ability to provide control for unstable processes and for systems without quantitative models [Coughanowr 65]. A typical closed-loop control system gathers information from sensors, calculates an error between the sensor value and a desired setpoint, and applies an offset based on this error. To be successful, these systems rely on both accurate and repeatable sensor signal processing, and on control based on models of the dynamics of the process being manipulated. However, accurate processing of sensor signals requires a model relating information from the sensor to the process under control; if the sensor is under development, it is not well understood and a complete model is frequently not available. If randomness is found in the sensor data, or conflicting indications are determined between sensors, these inconsistencies in data can propagate to induce oscillations in the process under control.

In addition, conventional programming environments do not support iterative prototyping and modification. These techniques are essential to the development of incompletely specified systems, since modifications must be applied to the overall system as information becomes available. The ability to deal with uncertainty is an important concept, since models are frequently incompletely defined; most conventional environments do not support this concept.

Commercially available AI Systems: Several systems or shells such as KEE™ and ART™ are available for the development of consultant type systems, also known as *expert systems*. These shells provide symbolic representation and inferencing, and heuristic search. Applications include Prospector, Mycin, and ADDAMX [Garcia 87]. The applications are frequently diagnostic, tutorial or predictive [Hayes-Roth 83] in style, and primarily provide off-line rather than on-line or real-time control. They are also commonly fragile in dealing with unanticipated conditions, not incorporating the concept of graceful degradation.

Real-time Control - Schemer: While incorporating the strengths of AI shells, such as symbolic representation and rapid prototyping, the Schemer architecture was also designed to respond to varying priorities of interrupts occurring in dynamic environments. Schemer is event-driven, and thereby provides appropriate response to multiple asynchronous interrupts. It provides the basis for adaptive problem-solving under time and resource constraints; in addition, it can incorporate various problem-solving techniques, both mathematically and heuristic based. Hence, it is appropriate for use in real-time control. While Schemer is an architecture, it has been implemented in several variations. The current implementation, discussed in this paper, was developed at Rockwell Palo Alto Labs (RPAL). In addition, research is on-going at RPAL both in terms of the architecture and additional implementations.

AROWS-Schemer: The specific application discussed in this paper is the enhancement of real-time control of the Advanced Robotic Welding System (AROWS) using Schemer in simulation mode. The existing real-time control system, based on conventional hardware and software, has been constrained by the lack of models of both the overall welding process and of the sensors. The sensors were developed in parallel with the implementation of the existing controller; no detailed models relating sensor data to the weld process are available. In addition, it is possible for the sensors to generate conflicting information. Due to these constraints and the complexity of the process, it was determined that Schemer should be applied in a feasibility study to determine the impact of an AI architecure on the operation of the real-time controller. In this manner, the initial concepts of the real-time controller may be enhanced without the restrictions inherent to the sensor development process; parallel efforts continue in sensor research. The use of Schemer for this application was found to be appropriate, both in the ability to respond to multiple interrupts from various sources and in the ability to iteratively develop, modify and augment the application in a clear manner. The details of this feasibility study and the effectiveness of Schemer in this particular environment are provided in subsequent sections.

Further directions, AROWS-Schemer: Additional future efforts with respect to this particular project could include interfacing with actual workcell components. While the simulated data has been generated in a manner to map closely onto actual sensor data, some unforeseen problems could arise in interfacing with specific hardware platforms or specific sensor systems. Schemer has been designed, however, to support the concept of graceful degradation, as opposed to many expert systems which are brittle in situations which exceed their expertise. The Schemer response to previously undefined states will be based on its ability to deal correctly with continuous data and its lack of dependence on explicitly stated rules.

SCHEMER BACKGROUND

Schemer Architecture

Schemer is a resource-bounded problem-solving architecture, with multiple implementations. The AROWS-Schemer application has been developed on a Common Lisp implementation currently running on Symbolics 3600 series systems, Macintosh II systems, and Xerox 1100 series systems. Schemer supports the interruption, suspension, and resumption of individual problem solving tasks, concepts which are essential to providing real-time response to changes in the environment.

Schemer has proven especially useful as the problem-solving architecture for systems that must perform satisfactorily in complex, dynamic environments. Successful Schemer applications have been built for a number of real-time, "process management" applications such as diagnosis or control of complex manufacturing processes [D'Ambrosio 87], automated performance management of advanced avionics systems [Guffey 86] and monitoring and task-management in a distributed information processing system [Fehling 84].

As shown in Figure 2, the Schemer architecture is similar to a blackboard architecture.

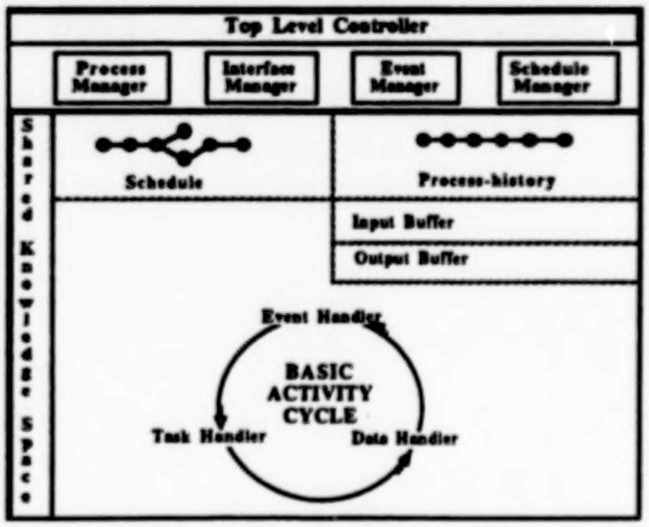


Figure 2: The Schemer Architecture:
A Top Level Controller providing prioritized interruptibility,
and Shared Knowledge Space for communication, containing Handlers, Schedule, and History

The architecture is comprised of three major components:

- Problem solving elements called Handlers or Procedural Elements (PEs), containing procedural and/or declarative knowledge;
- A global store, called Shared Knowledge Space, containing the Handlers as well as a current, prioritized Schedule of pending executable Handlers, and an audit trail, called the History;
- A Top Level Controller (TLC) that manages the system's activity.

The Shared Knowledge Space includes areas for communication with the environment (Input and Output Buffers), for communication of information between various problem solving elements (Handlers), and for state recording.

The overall control structure, the Top Level Controller, provides the prioritized interruptibility of the system. As indicated in figure 2, the TLC contains four Managers consisting of highly optimized code; the minimized execution time of a complete cycle provides maximum interruptibility. The TLC performs data transfer, determines which Handlers may be executed, maintains a variable-priority schedule, and causes Handler execution.

In the design of the Schemer architecture, every Handler is interruptible, and maintains its own local data space. The executable body of a handler may be a Common Lisp expression, a set of invocations of other Handlers, or an embedded Schemer.

This triad of major architectural components has been implemented in several instantiations. Following are further details of the specific implementation used for the AROWS-Schemer.

Schemer Implementation: TLC + PEs

This particular Schemer system implementation contains the basic elements interacting through a Shared Knowledge Space: the Top-Level Controller (TLC) which serves as the management structure, and Procedural Elements (PEs) which serve as the problem solving elements. The TLC controls execution of the overall process. The PEs include executable bodies and reference the Data Stores for inter-element communications and local state storage. The Shared Knowledge Space consists of the Schedule, a Data Store, and Ports.

The TLC consists of four managers, which repeatedly execute in the order indicated in figure 3. Since each manager consists of minimal code, the cycle responds rapidly to events in the environment, and in a prioritized manner. The PROCESS MANAGER executes the body of the first Procedural Element on the Schedule. The INTERFACE MANAGER queries all input / output (I/O) Ports to determine if any new data transfers are possible; it transfers all data available at input Ports to internal data storage areas, and transfers out-bound data from internal data storage to output Ports. The EVENT MANAGER, based on changes in internal Data Stores, determines if any Procedural Elements can be executed, i.e. whether the trigger conditions of any Procedural Elements have been satisfied, and enters these elements on the Schedule. The SCHEDULE MANAGER then executes any initializer code for new Procedural Elements on the Schedule, and orders the elements by priority. The cycle then repeats, with the PROCESS MANAGER executing the body of the highest priority Procedural Element on the Schedule.

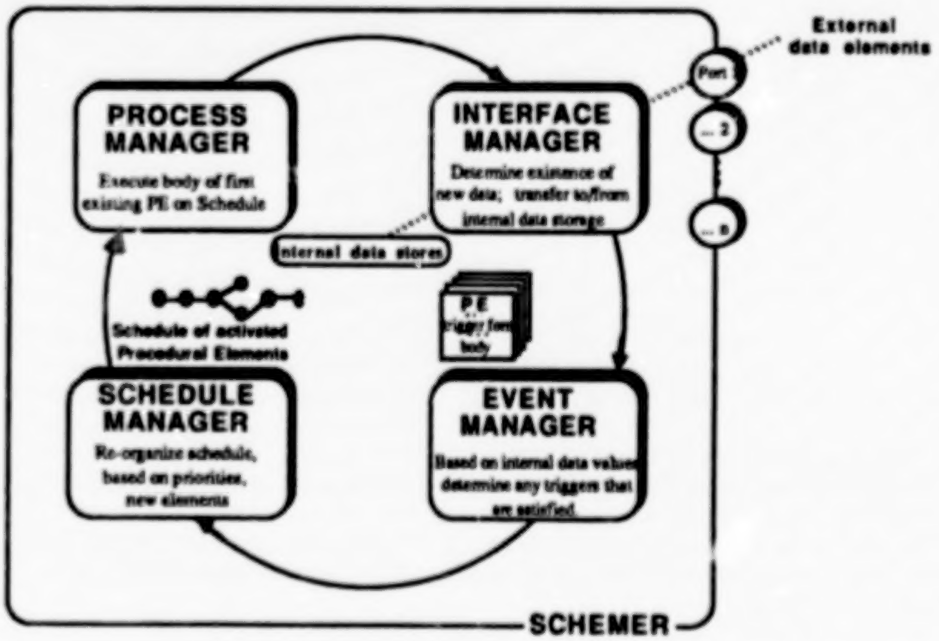

External data elements
Port 1
... 2
... n
PROCESS MANAGER
Execute body of first existing PE on Schedule
INTERFACE MANAGER
Determine existence of new data; transfer to/from internal data storage
internal data stores
Schedule of activated Procedural Elements
PE
trigger
body
SCHEDULE MANAGER
Re-organize schedule, based on priorities, new elements
EVENT MANAGER
Based on internal data values determine any triggers that are satisfied
SCHEMER

Figure 3: Overview of Schemer system implementation,
including cyclic execution of the four managers in the Top Level Controller
and the relationship of the Schedule, Procedural Elements, and external events.

Procedural Elements: Procedural Elements (PEs) are structures containing the Lisp or Schemer code to be executed based on the current state of the overall system. In addition to code, PEs also contain local variables, a list of relevant shared variables, and as appropriate, trigger conditions, initializing code, and priority levels. PEs may be activated in three manners: 1) their trigger conditions may be met, and they are scheduled by the Event Manager 2) another PE, during its execution, may explicitly schedule them for deferred execution, and 3) another PE may explicitly call them immediately, as a "sub-routine" to the original PE. In general, a PE includes a trigger definition, priority, and initialization code to be executed as it is placed onto the Schedule; PEs that are activated by other PEs need not incorporate all of these elements. Normally, PEs which must respond to asynchronous events in the environment are triggerable with high priority levels. PEs requiring lower priority treatment, frequently of lower time-criticality, are activated within triggered PEs. Possible elements of a PE are shown in figure 4, with sample values included.

PROCEDURAL ELEMENT STRUCTURE

component	sample values
BODY | INNER SCHEMER	(setf mod-current ae1)
[TRIGGER]	(> confidence-ae1 confidence-ae2)
[BASE-PRIORITY]	ordinary
[INITIALIZER]	()
SHARED-VARIABLES	(mod-current ae1 confidence-ae1 confidence-ae2)
PERSISTENT-VARIABLES	()
ACTIVATION-VARIABLES	()

Figure 4: Structure of a Procedural Element, including optional fields and sample values

Future Schemer Enhancements

Resource-bounded problem-solving requires that the system be aware of its current and past activities and its future commitments, as well as the relationship of these factors to conditions in the environment. However, in most realistic environments, the information available to the system is most frequently incomplete, and subject to change. The system must be able to deal with uncertainty in its knowledge of the world, and use what knowledge it has to bring about effective actions. An initial aspect to reaching a solution to incomplete information is the ability to respond to changes in the dynamic environment by the prioritized execution of specific tasks triggered by the specific state; the existing Schemer implementation incorporates this capability. An additional solution is the incorporation of the ability to deal with uncertainty, specifically by using a probabilistic decision-theoretic approach. Such an approach to control reasoning can prescribe how the problem-solving system can select among multiple, alternative problem-solving methods on the basis of how well each method satisfies the basic problem requirements, resource constraints, state of information, and the system's priorities and attitude toward risk. Mathematical decision-theory [Savage 72] can be used to provide a rigorous, verifiable and domain-independent basis for problem-solving control. Development is currently in progress to incorporate into Schemer the control of a system's actions based on decision analysis [Breese 88]; a basic mechanism is under development in the form of a set of general, domain-independent principles according to which the system controls and coordinates its actions under uncertainty.

Additionally, parallel research efforts are involved in modifying the TLC cycle, so that Managers are not necessarily executed in a defined loop. Rather, the Managers themselves will be event-driven. This supports implementation of the Schemer architecture in a parallel processor environment.

SPACE SHUTTLE MAIN ENGINE APPLICATION

The use of Schemer to enhance the existing real-time control system for the robotic welding of Space Shuttle Main Engines has been implemented as a simulation of the existing sensor controller tasks. The AROWS-Schemer application is a prototype system for simulation of both the robot motion and the weld process based on modification of pre-programmed values by multiple sensors. The existing application incorporates trend analysis, combination of sensor values over time and of cooperating sensors, use of confidence factors, and resolution of conflicting sensor data.

As indicated in figure 5, the application simulates and further enhances the actual work cell activities. That is, it accepts as initial input pre-programmed values for both the robot motion and the weld process parameter of current. During weld execution it coalesces data from multiple sensors to modify the pre-programmed values to more exactly match the actual variations in part geometry and material.

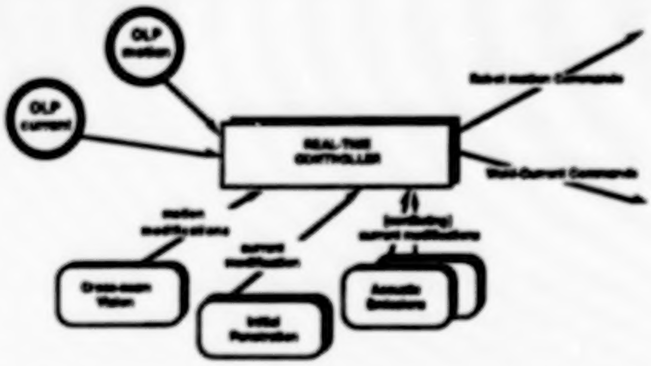


Figure 5: Pre-programmed motion and weld-process values are combined during weld execution with multiple sensor values.

As in the actual work cell, the real-time sensors are **vision** for cross-seam motion correction, and **penetration** sensors for weld-current correction [Gutow 87, Smith 87]. The Schemer simulation, however, is enhanced by the incorporation of multiple, interacting sensors. While the vision sensor is interleaved in time with the penetration sensors, there are both cooperative and conflicting penetration sensors. An **initial penetration** sensor monitors the weld until it determines that weld penetration has occurred, and modifies the pre-programmed weld-current to achieve initial weld penetration; at that time, the **acoustic emission (AE)** sensors begin penetration monitoring. Two AE sensors mounted at separate positions on the work piece generate possibly conflicting values which are inversely proportional to penetration; the pre-programmed weld-current value is modified by the AE value with the highest confidence factor.

The simulation is graphical, and uses lists of position and current values as initial input; it then responds to sensor values as they are entered asynchronously by the user (vision) or as simulated signals (penetration sensors). The output is graphical, indicating the corrected motion and current values. At this point in the feasibility study, no actual sensor data is used as input, and no formatted commands are generated for the workcell; at the time the system is implemented on a hardware platform more suited to a factory environment, these capabilities could be incorporated.

Progress Milestones

The initial goal of this project was determination of the feasibility of using Schemer to enhance the existing conventional sensor controller. The defined milestones included

- graphical simulation of a single process (motion) modified by a single sensor (vision)
- incorporation of multiple sensors modifying independent processes
    - the initial vision sensor modified the motion process

  - a generic penetration sensor modified the weld-current process parameter

  The two sensors did not interact in any fashion beyond co-existence.
- complex sensor interaction, including cooperating sensors, conflicting sensors, error checking, sensors affecting multiple processes

All milestones have been achieved; details of the milestone for complex sensor interaction are described in the following section. The Schemer environment was found to be appropriate for the development of this control system; it provides an environment for easy prototyping, partitioning of tasks, and response to changes in the dynamic welding process. Additional efforts include the incorporation of decision analysis techniques for quantitative model definition and use, further interaction with welding engineers for qualitative model definition, use of actual sensor-generated data, and implementation on a hardware platform suited to the factory environment.

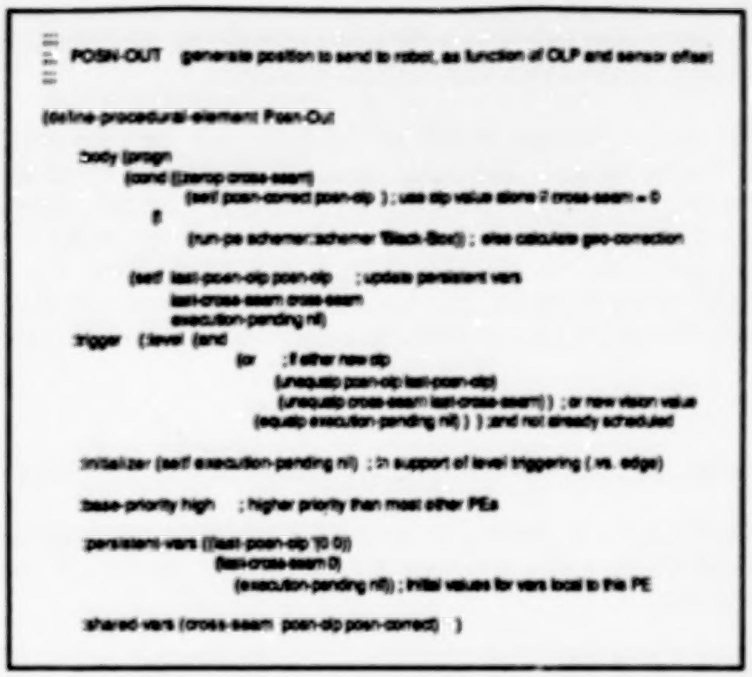
Demonstration of Complex Sensor Interaction, Affecting Multiple Processes

The Procedural Elements defined in the AROWS-Schemer are listed in table 1. In general, each sensor is simulated by a PE, and displayed on screen by use of implementation-specific monitor/window routines.

Procedural Element	Purpose
Time-Update	generate pseudo-run-time
OLP-Update	determine next Off-Line-Programmed (OLP) motion value
Posn-Out	combine OLP and vision sensor values, send to Robot; triggered by existence of new OLP or new vision value
Black-Box	called by Posn-Out to perform geometric calculations
Current-PE	combine OLP weld-current value with offsets from penetration sensors; triggered by existence of new penetration sensor values
AE-PE	generates new values from multiple AE sensors, including confidence factor based on simple trend analysis
Initial-Pen	generates modifications for weld-current until penetration occurs

Table 1: Procedural Elements of AROWS-Schemer and their purpose

An example of a specific PE is shown in figure 6. This PE performs the function of combining the appropriate Off-Line-Programmed position value with the most recently determined vision sensor value for cross-seam correction. It initiates execution of another PE ("Black-Box") for the actual geometric calculation, and is triggered by the existence of either a new OLP value or a new vision cross-seam value.



```
;;;
;;; POSN-OUT    generate position to send to robot, as function of OLP and sensor offset
;;;

(define-procedural-element Posn-Out

   :body (progn
           (cond ((zerop cross-seam)
                    (setf posn-correct posn-olp ) ; use olp value alone if cross-seam = 0
                 (t
                    (run-pe schemer:schemer 'Black-Box)) ; else calculate geo-correction

           (setf last-posn-olp posn-olp     ; update persistent vars
                 last-cross-seam cross-seam
                 execution-pending nil)
   :trigger   (:level (and
                       (or     ; if either new olp
                           (unequalp posn-olp last-posn-olp)
                           (unequalp cross-seam last-cross-seam) ) ; or new vision value
                         (equalp execution-pending nil) ) ) ;and not already scheduled

   :initializer (setf execution-pending nil)  ; in support of level triggering (vs. edge)

   :base-priority high     ; higher priority than most other PEs

   :persistent-vars ((last-posn-olp '(0 0))
                     (last-cross-seam 0)
                        (execution-pending nil)) ; initial values for vars local to this PE

   :shared-vars (cross-seam  posn-olp posn-correct)   )
```

Figure 6: An example Procedural Element, Posn-Out

which runs at high priority to insure the robot always has the most recent corrected position value.

Conflict Resolution: There are several situations where conflicts may arise between sensors. In general, similar sensors monitoring a process may provide conflicting information about that process based on sensor irregularities, or dissimilar sensors may generate conflicting data due to variations in processing algorithms and their applicability to a specific task. There are several strategies which could be used to resolve the conflict: one sensor could be designated as the primary sensor, with the secondary sensor data used only when process history indicates invalidity of the primary sensor; world models could be used, including trend analysis of sensor data as well as the specifics of the task; or sensor determined confidence factors could be compared.

In the AROWS-Schemer, two Acoustic Emission (AE) sensors are simulated, each generating independent data. The choice of correct weld-current modification must be made between the two independent values at each time slice, as time progresses under control of Time-Update. Both sensors are treated at equal priority level, and simple trend analysis is used to generated confidence factors in each set of data, as indicated by equation (1).

$$\begin{aligned}\text{confidence} = 100 - {} & \text{abs}\,[\,\Delta \text{ this sensor's previous \& current values}\,] \\ & - \text{abs}\,[\,\Delta \text{ current value and last applied value}\,] \qquad (1)\end{aligned}$$

The AE with the higher confidence factor is then applied to the pre-programmed weld current. As calculated, the confidence factor tends to minimize the variations in the applied value of the AE sensors, i.e. the sensor value that is most consistent with both its previous value and with the last applied value is chosen. Hence, the pre-programmed values receive a higher weighting factor than "noise" in the sensor values. This is an ad hoc method of defining confidence, with no experimental basis. A mathematically based technique that will provide a more formal basis is described in the next section.

Examples of the application during execution are shown in figures 7-9, as time progresses. The initial pre-programmed values are indicated by dark circles and lines for both robot motion (a circular overlay weld) and weld-current (three pre-programmed levels are indicated). During execution, the user may generate varying vision sensor values by keyboard input; these are applied, after error checking, to the robot-motion values as cross-seam offsets. The corrected values are indicated by clear circles and dashed lines. Concurrently, at the beginning of the weld, an

ORIGINAL PAGE IS
OF POOR QUALITY

Initial-Penetration sensor determines the correction to be applied to the weld-current. Penetration sensor values are combined over time before being applied as offsets to the pre-programmed weld-current values, yielding a smoothed correction to the initial values. Once weld-penetration has occurred, the AE sensors become active for monitoring penetration and modifying weld-current to maintain penetration. The AE value which is actually applied is indicated in the AE screen area by darkened circles.

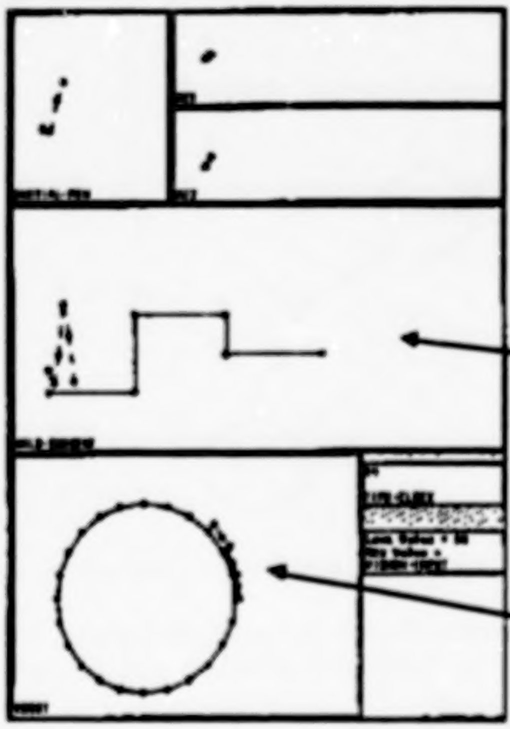
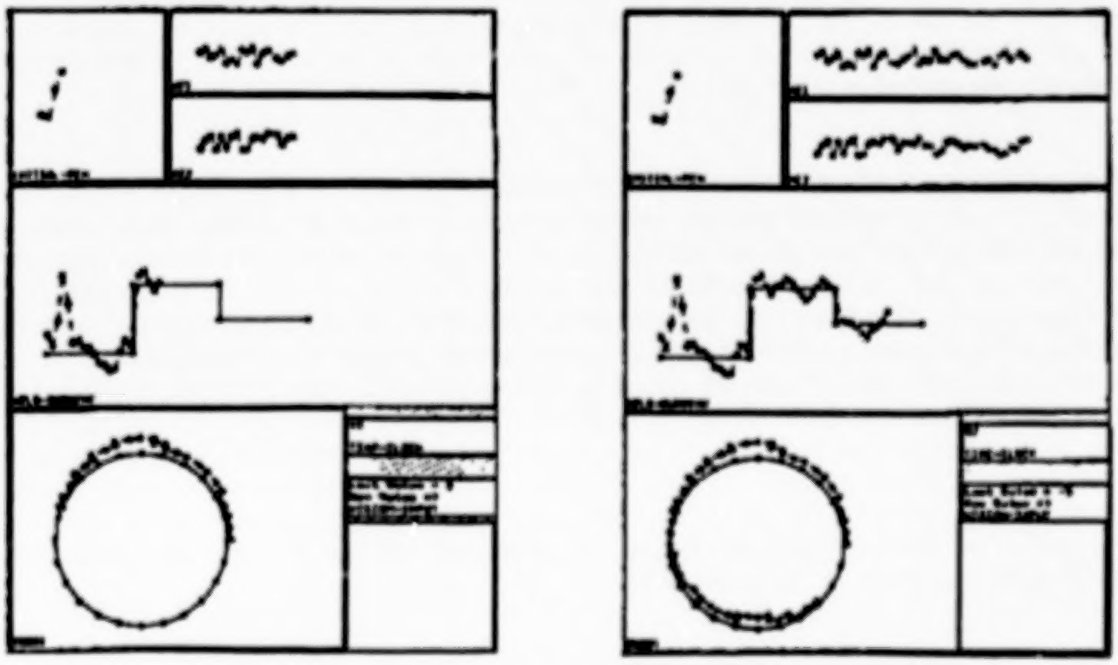


Figure 7: Simulation screen dump, at time = 14.

Initial penetration sensor has just completed control of current, and AE sensors now begin to compete.

Vision sensor has modified the pre-programmed path, as indicated by clear circles.



Figures 8,9: Additional screen dumps during execution of simulation, taken at time = 46, 87, including more acoustic emissions and vision data

Decision Analysis Techniques: Although not yet incorporated in the Schemer architecture, decision analysis methods are being applied toward this application. Influence diagrams are used to represent the model upon which decisions are based; the principles explicitly represent and respond to uncertainty and incomplete information [Breese 88]. An influence diagram has been defined for the use of the vision sensor, and probabilistic relations have been defined. At the time of this writing, no automated application of the influence diagram has been incorporated. It is expected that the influence diagram will be used both in the initial model definition, i.e. as an input to the AROWS-Schemer demonstration, and in real-time decision making within Schemer. An initial influence diagram is shown in figure 10.

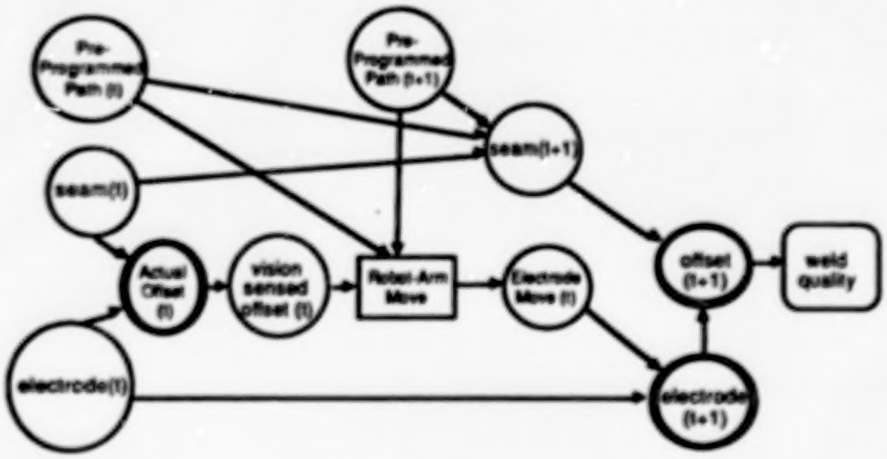


Figure 10: Influence Diagram for the vision sensor affected pre-programmed motion values

The influence diagram represents the information known at the time a decision is made, for example "Robot Arm Move" is based on both the pre-programmed values and the vision sensor value. The vision sensor value is affected by the actual offset of the electrode from the seam, as well as the quality of the vision sensor itself, in terms of the general seam being processed and with respect to a recent set of images acquired. The overall quality of the weld, with respect to position, is determined by the offset of the electrode from the actual seam. All of these relationships can be defined in a probabilistic manner, and solved for the optimum decisions, providing a mathematically based model for the control of the workcell.

Future efforts: While the existing system uses keyboard input for the vision sensor, value could be gained by using a filtered random number generator to create the vision sensor values. In this mode, additional signal processing and trend analysis could be used as the basis of geometric plan transformation, depending on the specific task and the trends detected in the sensor values. Influence diagrams are also under development for representation of the complete set of work cell activities, that is the dependence of weld quality on penetration as well as position accuracy. In this case, the detailed interaction of sensors could be simulated and a more accurate model of the decision processes could be developed. Finally, while the current implementation is on a Symbolics 3600 series system in Common Lisp, the Schemer implementation itself ports to any hardware platform supporting Common Lisp. Efforts are continuing to determine an appropriate system, and are influenced by the capabilities of existing conventional systems in the work cell. When an appropriate platform has been determined, additional efforts could include actual sensor data acquisition as well as command generation.

SUMMARY

A prototype system has been described which successfully applies a resource-bounded problem-solving architecture to simulated real-time control of robotic welding of Space Shuttle Main Engine (SSME) assemblies. The Schemer architecture supports real-time response to multiple sensor interrupts, both cooperative and conflicting. It is event-driven and responds to varying priorities in an adaptive manner; the implementation is general, and can be ported to platforms supporting Common Lisp. Schemer can be used in support of real-time control in both space-based and manufacturing environments, and provides the basis for prioritized dynamic response to changing environments, as well as the ability to incorporate mathematically based techniques from the field of decision analysis. The Schemer architecture was found to be appropriate for the development of an SSME manufacturing control system; it provides an environment for easy prototyping, partitioning of tasks, and response to changes in the dynamic welding process. The SSME application exemplifies the strengths of this knowledge-based architecture over conventional architectures, especially in the ability to respond immediately and in a prioritized manner to changes in a dynamic processing environment and the ability to apply both mathematically and heuristically based knowledge to a real-time activity.

ACKNOWLEDGEMENTS

The work described in this report was funded under NASA contract NAS8-40000 mod 130. The author is indebted to members of Rockwell Science Center, Palo Alto Laboratory: to Michael R. Fehling and B. Michael Wilber for their definition and implementation of the Schemer architecture, to Jack Breese for his direction in decision analysis techniques, to Greg Arnold for his expertise in the existing sensor controller efforts, and to Art Altman for his editing support. In addition, appreciation is extended to Rocketdyne personnel Matthew A. Smith and David Gutow for their exisiting work in acoustic emission and vision sensors, respectively; to Joseph M.F. Lee of Rockwell Science Center as Project Manager; and to Fred Schramm as NASA Contract Monitor.

REFERENCES

[Breese 88a] Breese, John S., Eric J. Horvitz, and Max Henrion, "Decision Theory In Expert Systems and Artificial Intelligence", Technical Report 3, Rockwell International, 1988

[Breese 88b] Breese, John S. and Michael R. Fehling, "Decision-Theoretic Control of Problem Solving: Principles and Architecture", Proceedings AAAI Workshop on Uncertainty in Artificial Intelligence, 1988

[Coughanowr 65] Coughanowr, D.R., Process Systems Analysis and Control, McGraw-Hill, 1965

[D'ambrosio 87] D'Ambrosio, Bruce, M.R.Fehling, S.Forrest P.Raulefs, and M. Wilber, "Real-Time Process Management for Materials Composition in Chemical Manufacturing", IEEEExpert, June 1987

[Fehling 87] Fehling, Michael R. and John S. Breese, "A Computational Model for the Decision-Theoretic Control of Problem Solving under Uncertainty", Technical Report, Rockwell International, 1988

[Guffey 86] Guffey, J., "AI takes Off: Expert Systems that are Solving In-Flight Avionics Problems", Aviation Week and Space Technology, February 1986

[Garcia 87] Garcia, Raul C., "An Expert System To Analyze High Frequency Dependent Data for the Space Shuttle Main Engine Turbopumps", Proceedings AI for Space Applications, 1987

[Gutow 87] Gutow, David, Member Technical Staff, Advanced Automation and Robotics, Rocketdyne, Private communications

[Hayes-Roth 83] Hayes-Roth, Frederick, Donald A. Waterman, Douglas B.Lenat, Building Expert Systems, Addison-Wesley, 1983

[Ruokangas 88] Ruokangas, Corinne C., B.M.Wilber, D.L.Larner, and M.C.Anderson, "Schemer: A User's Guide", Technical Note, Rockwell Palo Alto Lab,1988

[Savage 72] Savage, L.J. The Foundation of Statistics, Dover:New York, 1972

[Sliwinski 87] Sliwinski, Karen E. and Corinne C. Ruokangas "Adaptive Robotic GTA Welding for the SSME: An Integrated System", Conference Proceedings Robots 11, 1987

[Smith 87] Smith, Matthew A., Member Technical Staff, Advanced Process Instrumentation, Rocketdyne, Private communications

[Winston 87] Winston, Patrick H., "Artificial Intelligence: A Perspective", AI in the 1980's and Beyond, MIT Press, 1987

N89-15552

Solutions to Time Variant Problems of Real-Time Expert Systems*

Show-Way Yeh and Chuan-lin Wu
Department of Electrical and Computer Engineering
The University of Texas
Austin, TX 78712

Chaw-Kwei Hung
Information Division
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, Ca 91109

ABSTRACT: Real-time expert systems for monitoring and control are driven by input data which changes with time. One of the subtle problems of this field is the propagation of time variant problems from rule to rule. This propagation problem is even complicated under a multiprogramming environment where the expert system may issue test commands to the system to get data and to access time consuming devices to retrieve data for concurrent reasoning. There are two approaches which have been used to handle the flood of input data. Snapshots can be taken to freeze the system from time to time. The expert system treats the system as a stationary one and traces changes by comparing consecutive snapshots. In the other approach, when an input is available, the rules associated with it are evaluated. For both approaches, if the premise condition of a fired rule is changed to being false, the downstream rules should be deactivated. If the status change is due to disappearance of a transient problem, actions taken by the fired downstream rules which are no longer true may need to be undone. If a downstream rule is being evaluated, it should not be fired. Three mechanisms for solving this problem are discussed in this paper: forward tracing, backward checking, and censor setting. In the forward tracing mechanism, when the premise conditions of a fired rule become false, the premise conditions of downstream rules which have been fired or are being evaluated due to the firing of that rule are reevaluated. A tree with its root at the rule being deactivated is traversed. In the backward checking mechanism, when a rule is being fired, the expert system checks back on the premise conditions of the upstream rules that result in evaluation of the rule to see whether it should be fired. The root of the tree being traversed is the rule being fired. In the censor setting mechanism, when a rule is to be evaluated, a censor is constructed based on the premise conditions of the upstream rules and the censor is evaluated just before the rule is fired. Unlike the backward checking mechanism, this one doesn't search the upstream rules. This paper explores the details of implementation of the three mechanisms.

1. Introduction

A real-time expert system is data-driven and is different from the conventional expert systems. For a conventional expert system, the premise conditions of rules do not change with time during the entire transaction, and the reasoning is directed by the operator. Establishing proper communication protocols between the expert system and the sensors and processors in the underlying system enables the expert system to monitor the

---

* This work is supported by Jet Propulsion Laboratory under contract GK857812.

underlying system and issue commands to control the underlying system directly. For a real-time expert system, the reasoning is triggered by the input data which are collected from the underlying system it controls [Cot87, Laf88, Lei87, Moo86, Pat85, Sau83]. The premise conditions of rules are predicates on the input data and on conclusions of other rules. Since the input data vary with time, the premise conditions of the real-time expert system rules vary accordingly. If a premise condition of a rule includes a predicate on the conclusion of another rule, then, if the premise condition of the latter is changed, the premise condition of the former will be affected because the conclusion of the latter is changed. That is, if the firing of a rule depends on another rule, then the changing of the premise condition of the latter will propagate to the former. This time variant propagation will continue if the conclusion of the former is a premise condition of other rules.

The input data come to the real-time expert system from processors or sensors distributed in the underlying system. Since the data vary with time, the real-time expert system also needs to reason in a time variant manner. The real-time expert system may take snapshots of the underlying system periodically. That is, it freezes the underlying system from time to time. Changes are traced by pairwise comparison of consecutive snapshots. The real-time expert system reasons using the input data from each snapshot and the changes between every two consecutive snapshots. If any changes are detected, the real-time expert system starts to investigate whether there are any problems. There is another mechanism in which, when an input data item is available, the system immediately investigates what happened and, if something has changed, takes actions if necessary.

This paper discusses problems of the time variant property of real-time expert systems. Although the underlying system addressed in this paper is assumed to be a real-time network system and all examples are based on network management, the problems are common to real-time monitor and control expert systems in all the various other application areas. The reader can easily convert the analogous problems to those of other real-time monitor and control expert systems. Section 2 describes the time variant problems of the real-time expert systems in detail and defines some of the terms we are using. Section 3 presents the reasons why a multiprogramming environment is necessary for some real-time systems. In a multiprogramming environment, a concurrency control mechanism is necessary to fire and to stop the firing of rules. Three mechanisms of deactivating rules being evaluated, forward tracing, backward checking, and dynamically censor setting, are discussed in section 4, 5, and 6. A comparison of these three mechanisms are presented in section 7. Finally, a brief conclusion is given in section 8.

2. Time Variant Problems

If any premise condition of a rule is a part of the conclusion of another rule, then the former is called a downstream rule of the latter and the latter is called an upstream rule of the former. If X is a downstream rule of Y and Y is a downstream rule of Z then X is called a downstream rule of Z and Z is called an upstream rule of X.

When the premise conditions of a fired rule become false, all downstream rules which have been fired or are being evaluated on the basis of the firing of that rule need to be reevaluated. For example, suppose that we have the following four rules:

if A then B and F
if B and C then D
if D then E
if F or G then H

Suppose that A becomes true at time T for a short period of time and C is true after time T as well. Then the above four rules should be fired sequentially after T. Suppose that by the time the rule *if B and C then D* is being evaluated A becomes false. Then all four of these rules should not be fired. For the already fired rule, *if A then B and F,* since actions of B and F have been done, complementary actions may need to be performed. For the rule being evaluated, *if B and C then D,* evaluation should be stopped and the rule should not be fired. The rule *if D then E* doesn't need to be evaluated.

If delay is tolerable, a transient time interval may be defined for each rule, and a rule is to be fired when the premise conditions remain true longer than the predefined interval [Lei86]. However, if there is not enough time available to make sure that the problem persists, actions should be taken place immediately and complementary actions should be taken if the problem is disappears. For example, if the real-time monitor and control expert system of a network receives a report saying that a command packet keeps being refused by another node and the packet must be received by the destination spacecraft in a very short time, the expert system may send commands to the sender to change the routing table to send the packet along the secondary path to the destination to meet the time limit constraint. However, the reason for refusal to receive packets may be that the input buffers of the receiver are temporarily full because too many nodes are sending packets to it at the same time. After the expert system sends the changing command to the sender, it may receive a report saying that the problem is gone. So, the expert system must send another command to the sender to change the routing table back to the original one to keep the traffic balanced.

Also, when an error has been corrected, the fired rules which managed the error should be deactivated. For example, if a node is dead, the routing tables of other nodes need to be changed to bypass the error. Once the node has been fixed, the routing tables need to be changed back to the original ones.

For all these cases, rules which have been fired need to be deactivated and evaluation of rules needs to be stopped when the premise conditions of upstream rules which resulted in firing or evaluation of the rules are changed to being false.

3. Execution Environment

Since the input data change with time, the actions to be taken will vary accordingly from one time to another. To maintain the whole system, changes in input data and actions taken in the past may need to be recorded somewhere in the system so that the expert system may trace what happened to the underlying system in the past. For example, suppose that each node in a network records the inbound and outbound packet headers in the network management database. Suppose that the expert system is reported that a packet is lost. Let C of the rule *if B and C then D* be the predicate of evaluation of recorded headers in the file storage of remote nodes. When the expert system evaluates $C,$ it will send commands to all nodes along the path of the lost packet to retrieve the history headers to determine where is the problem. Before the data returns, the evaluating process can do nothing but waiting.

Meanwhile, the real-time expert system may need some current data from the underlying system to make decision. It may issue commands to processors or sensors in the underlying system to perform certain tests. For example, the rule *if A then B and F* is to determine that a node has generated noise. Suppose it is fired. Futhermore, let C of the rule *if B and C then D* be the predicate to evaluate the test value of the interface card of the node. The expert system will issue a command to the troublesome node to perform the test. The expert system will make decision based on the test result.

Retrieving history data from remote data bases and instructing certain processors or sensors to perform tests usually are time consuming. When the expert system needs to do it, it cannot continue processing anything. On the other hand, while the expert system is waiting for the remote data or test results, the expert system cannot stop to wait because the underlying system status is changing and the flood of input data continues to arrive.

To process the new data and to manage new problems while the expert system is waiting, new processes need to be created. Therefore, expert systems of this kind must be implemented in a multiprogramming environment. A process is suspended when it needs to wait for information which is not currently available in the local system. It is resumed when the information for which it is waiting becomes available.

The time variant problems are even worse in the multiprogramming environment. For example, while a process is evaluating the rule *if B and C then D,* the input data associated with *A* may be changed by another process such that *A* becomes false. The latter, after deactivating the rule *if A then B and F* will find that, since B has been changed to be false, the rule *if B and C then D* needs to be reevaluated. So, two processes may not know each other and work independently and concurrently. They may evaluate the same rule and get different conclusions. Finally, conflict actions may be taken.

For convenience, we define the following terms. A process which evaluates the premise condition of a rule which is not fired and fires that rule if the premise condition becomes true is called an activating process. When input data are changed such that the premise condition of a rule becomes true, the rule will be fired and the conclusion will be committed to be true. If the commitment makes the premise condition of another downstream rule to be true, an activating process will fire the downstream rule, too. So, the activating process will propagate to the downstream rules until no downstream rules can be fired. On the other hand, A process which evaluates the premise condition of a rule which has been fired and deactivates that rule if the premise condition becomes false is called a deactivating process. When input data are changed such that the premise condition of a fired rule becomes false, the rule will be deactivated, complementary actions will be taken, and the conclusion will be committed to be false. If the commitment makes the premise condition of another fired downstream rule to be false, a deactivating process will deactivate the downstream rule, too. So, the deactivating process will propagate to the downstream rules until no fired downstream rules need to be fired.

If the rule *if F or G then H* is not fired and two activating processes evaluate *F* and *G* concurrently and respectively, then both processes will find that the premise condition becomes true and the rule will be fired twice. If the actions are expensive, for example, if *H* is to issue commands to some processors to perform certain tests, we want the actions to be taken only once. This problem can be easily solved by write-locking the firing status of each rule when an activating process checks to see whether the rule is fired or not. Similarly, if the rule *if B and C then D* has been fired and two deactivating processes evaluate *B* and *C* concurrently and respectively, then both processes will find that the premise condition becomes false and the rule will be deactivated twice. If the complementary actions are expensive, we want that the complementary actions are taken only once. This problem can be easily solved by write-locking the firing status of each rule, too.

Comparing a deactivating process with an activating process, we can find they are similar. When a deactivating process evaluates the premise condition of a fired rule, if the complement of the premise condition is true, it deactivates the rule and takes complementary actions. To deactivate a rule is just like to "fire" a rule to take the complementary actions. For example, the complement rule of the rule *if A then B and F* is *if A(past) and not A(now) then (not (B and F))* or *if (B and F) and not A then (not (B and F))*. To deactivate the first rule is the same as to activate either the second rule or the third rule.

However, since the expert system is to monitor and control the underlying system, the rules are to handle problems. If an activating process fires rules, most likely, the associated actions will lead the underlying to abnormal state to avoid the problems. In contrast with the activating process, when a deactivating process deactivates a fired rule, it usually means that the problems managed by the fired rule are gone. The associated complementary actions are to lead the underlying system back to normal state. Usually, the running cost of the underlying system in a normal state is lower than that in an abnormal state. Therefore, if the problems are transient, we prefer that the deactivating processes have higher priority to "catch up" the associated "front runing" activating processes to stop them to keep the system in the normal states as much as possible. That is, we want that when a rule is fired, it is consistent with the current status of the input data. Three mechanisms, forward tracing, backward checking, and dynamic censor setting, of doing so are discussed.

4. Forward Tracing

In forward tracing, we just let the deactivating processes have higher priority than the activating processes. When the premise conditions of a fired rule become false, the premise conditions of the downstream rules which have been fired or are being evaluated due to the firing of that rule are reevaluated. The fired downstream rules are deactivated and the downstream rules being evaluated are stopped if the premise conditions become to be false. The deactivating process will traverse a tree rooted at the rule whose premise condition has just been changed to being false. Each leaf node is the first downstream rule which has not been fired or has been fired by other rules.

To deactivate a fired rule, the complementary actions are dependent on the actions of the rule. For example, if the action is to use alternative routing table, the complementary action is to use the principal routing table; if the action is to double the time-out period, the complementary action is to use the original time-out. There are not systematic ways to develop the complementary actions. They need to be developed rule by rule.

For the rules used as examples in the previous sections, when the rule *if B and C then D* is evaluated, *B* may be read into a working buffer. Then, when *A* becomes false, since the deactivating process has higher priority, *B* may be changed to being false before the activating process finishes evaluating the rule. The activating process acting on this rule may not know that *B* has been changed to being false and continues to evaluate the rule. Consequently, this rule and the rule *if D then E* will be fired.

It is apparent that concurrency control mechanisms are needed to make sure that changes of input data can affect the evaluation of the rules being evaluated. The traditional locking mechanisms may not be adequate to solve this problem because the deactivating process is designed to update data which have been read by the activating process. If the traditional locking mechanisms are applied, the activating process will read-lock the items it reads. This prevents the deactivating process from updating the items they have been locked. The deactivating process can never "catch up" with the "front running" activating processes. Since the deactivating process has higher priority than the activating process, the traditional locking mechanisms are not applicable for the forward tracing mechanism.

Soft lock [Ege86, Yeh87] is a locking mechanism for software design data base systems. Since a designer may spend a long time figuring out how to design or update an item which can be a requirement, a specification, a circuit design, or a module of code, for each item there may be several versions. The designer may wish to reference others' work while he is doing his own. If the traditional locking mechanisms are applied to these systems, no one can update an item which is being referenced by others and no one can reference an item which is being updated by others. This will block designers from continuing with their work. Soft lock is designed to set a conflict mark on an item when a conflict between two transactions is detected. The two designers will be informed either as soon as the conflict is detected or at the end of the transaction that ends sooner. Then, the designers will resolve the conflict.

Similarly, we may allow the deactivating process to break a soft lock set by the activating process. When the activating process starts to evaluate a rule, it softly locks the premise conditions and then reads them. When the deactivating process wants to change any premise condition of a rule, it checks first whether a soft read lock has been set on the premise condition. If yes, it breaks the lock and updates the value. Finally, when the activating process finishes evaluating the rule, it checks to see if any soft locks have been broken. If yes, it reevaluates the rule. Otherwise, it fires the rule and unlocks the locks.

For the OR rule *if F or G then H,* when A is true, F is true. No matter whether G is true or false, this rule will be fired. But, G may become true at any time. If, when A becomes false, G has become true, this rule will not be deactivated and is a leaf node of the tree which is being traversed by the deactivating process.

Obviously, there is no guarantee that all deactivating processes can "catch up" with the activating processes.

5. Backward Checking

To overcome the drawback of forward tracing that the deactivating process may not be able to "catch up" with the associated activating processes acting on downstream rules before the rules are fired, the expert system may check back on the premise conditions of all upstream rules to see whether any of them have been changed right before the rule being evaluated is fired. Backward checking means that when a rule is being fired, the expert system checks the premise conditions of the upstream rules which result in evaluation of the rule to see whether they are still true.

For the rules used as examples in the previous sections, when the rule *if B and C then D* is finished being evaluated, the expert system checks back to see whether B and C are still true. Since B is produced from A being true, the expert system needs to check whether A and C are still true. That is, a tree rooted at the rule being evaluated is traversed. Similarly, when the rule *if D then E* is finished being evaluated, the expert system checks back to see whether D is still true. Since D is produced from B and C being true and B is produced from A being true, the expert system needs to check back to see whether A and C are still true.

For the rule *if F or G then H* if only *F* or *G* changes to being false and the other remains true, the rule cannot be deactivated. Since *F* is produced by the rule *if A then B and F,* the backward checking mechanism needs to check the premise conditions of both rules. Only if both *A* and *G* become false should the rule be deactivated. If only one of *F* and *G* is true and the other is false or unknown when the rule is evaluated, then the checking back mechanism only checks the upstream rules of the predicate whose truth value is true.

The rule is fired only if its premise conditions are still true. If there is not enough time to check back, the rule is fired without checking back. Since the forward deactivating process and the backward checking process head toward each other, they will meet each other somewhere in the middle. This guarantees that if a premise condition is changed, all downstream rules which are being evaluated will not be fired if the change will deactivate them.

The locking mechanism needed for the backward checking mechanism is similar to that for the forward tracing mechanism. The backward checking activating process will softly lock the premise conditions of an upstream rule before it starts to evaluate the premise conditions. The deactivating process can break soft locks to update the premise conditions. The locks on the rules which have been fired can be traditional locks or soft locks.

6. Dynamically Setting Censors

In the backward checking mechanism, when a rule is evaluated, all premise conditions of the upstream rules need to be checked. A drawback of this mechanism is that the upstream rules need to be searched and searching may be expensive.

Variable precision logic rules have the form *if X then Y unless Z*. If, when *X* is true, *Y* is true with a high probability x and is false only if *Z* is true with low probability 1-x, then, if there is not enough time to evaluate Z, we may fire the rule *if X then Y*. The probability of getting the correct result is high. Z is called a censor [Mic86, Had86].

Dynamically setting censors means that, when a rule is to be evaluated, a censor is constructed based on the premise conditions of the upstream rules and the censor is evaluated just before the rule is fired. For the rules in the previous sections, suppose *A* and *C* are predicates on input data and are true. The rule *if A then B and F* will be evaluated and be fired. When it is being evaluated, the rule is reconstructed as *if A then B and F unless (not A).* This rule will not be fired if *A* becomes false while the rule is being evaluated. If *A* continues to be true and this rule is fired, since *B* becomes true, the rule *if B and C then D* should be fired. Therefore, since what may be changed are *A* and *C,* before we evaluate the rule, the rule is reconstructed to be *if B and C then D unless (not A or not C).* Then, if *A* or *C* changes to being false while this rule is being evaluated, this rule will not be fired. Similarly, when we are to evaluate the rule *if D then E* is to be evaluated, since what the activating process needs to check is *A* and *C,* we reconstruct the rule to be *if D then E unless (not A or not C).* It will not be fired if *A* or *C* becomes false while it is being evaluated.

The censor looks awful if the premise condition of the rule is AND of predicates because all of the predicates are taken into the censor. If the premise condition of a rule is inclusive OR of predicates, then the censor is the complements of the predicate which makes the rule to be fired. This is because that the censor is to prevent from transient problems. When the rule is about to be fired, what we worry about is whether the premise conditions of the upstream rules have been changed such that the current rule should not be fired. For example, if F is changed to being true and the truth value of G is unknown, the rule *if F or G then H* should be evaluated and to be fired. But, before evaluating it, it is reconstructed to be *if F or G then H unless (not A)* where the censor is inherited from the rule *if A then B and F* by F. So, when the activating process finds that this rule should be fired, it checks to see whether A is still true. If yes, fires the rule. Otherwise, it stops itself.

This mechanism only evaluates predicates on input data. The upstream rules are not evaluated and are not searched. When evaluating a rule results from firing other rules, it is easy to construct the censor. It is the combination of the censors of the upstream rules and the input data of the current rule. For example, the premise of the first rule, A, is a predicate on the input data. Its censor is the complement of the predicate. For the second rule, B is a predicate which is the conclusion of the first rule. The censor of the first rule is inherited. C is a predicate on the input data. So, the censor is the inclusive OR of the inherited predicate from the first rule and the complement of C. For the third rule, the premise predicate is the conclusion of the second rule. So, the censor of the second rule is inherited to be the censor of the third rule. For the fourth rule, since the premise condition is the inclusive OR of two predicates, the censor is the complement of the one which makes the rule be evaluated. If the predicate is conclusion of an upstream rule, the censor is inherited from the upstream rule.

Some of the problems can never be transient problems. For example, if a packet is reported to be lost by the operator, this problem cannot be gone in a short time. Another kind of problem, for example, error rate of a link is detected to be a little bit too high 10 times in an hour. If it is detected only once, that may be tolerable. But, the problem is detected by counting how many times it happened. Once the number is higher than the threshold, the problem is identified. To count the number of times, once the high error rate is detected, it will counted once and not be changed. For this kind of problem, since no transient problem, censor is not necessary. That is, censor is set only when the input data do be able to be changed transiently.

7. Comparison of the Three Mechanisms

The simplest one among the three mechanisms is the forward tracing mechanism. Since it doesn't guarantee that the activating processes can be caught up by the associated deactivating processes when input data of upstream rules are changed to being true transiently and then are changed to being false, if the cost of firing the rule is expensive, this mechanism should not be used. The other two mechanisms guarantee that if the input data of upstream rules are changed to make the rules false while the rule is being evaluated, the activating process will be stopped. The backward checking mechanism needs to search the upstream rules to reevaluate the predicates. It not only reevaluates the input data associated with the rule being evaluated, but also reevaluates the intermediate predicates. While the dynamic setting censor mechanism only reevaluates the input data and doesn't need to search for the intermediate upstream rules. So, the backward checking is less efficient than the dynamic setting censor mechanism. If the actions of a rule are very expensive and the rule needs to be evaluated in very short time, dynamic setting censor mechanism is the best choice.

8. Conclusion

In this paper, we address some characteristics of real-time monitor and control expert systems. The input data are collected from the underlying system and are changing with time. When the input data change, the rules which have been fired or are being fired on the basis of the changed data need to be reevaluated. When the premise condition of a rule is changed, the downstream rules need to be reevaluated, too. That is, the time dependent changes may propagate from rule to rule. If a fired rule needs to be deactivated, complementary actions need to be taken to overcome the change. If the premise condition of a rule that is being evaluated is changed, the activating process needs to be stopped. This time variant problem gets worse in the multiprogramming environment. Three mechanisms are discussed in this paper. In the forward tracing mechanism, the deactivating process has higher priority than the activating process. This mechanism cannot guarantee that the deactivating process can catch up with the activating process. To overcome this drawback, a backward checking mechanism may be used that, when a rule is about to be fired, checks the upstream rules to see whether the rule still should be fired. The drawback of this mechanism is that extra searching is needed and is expensive. In the censor setting mechanism, changes of input data are inherited from rule to rule to eliminate the searching overhead of the backward checking mechanism. Since the backward checking and the dynamically setting censor mechanisms need some redundant work, the rules need to be investigated one by one to determine whether it has transient problem. If not, these two mechanisms should not be applied.

Reference

[Cot87] Cotthem, H., Mathonet, R., and Vanryckeghem, L., "Dantes--An Expert System Shell Dedicated to Real-Time Network Troubleshooting", Workshop on Expert Systems and Network Operation, IEEE, ICC, 1987.

[Dvo87] Dvorak, D., "Expert Systems for Monitoring and Control--A Survey", University of Texas at Austin Technical Report AI87-55, May 1987.

[Ege86] Ege, A., Ellis C., and Wexelblat, A., "Design and Implementation of GORDION, An Object Base Management System", MCC Tec. Rep. No. STP-139-86(Q), April 1986.

[Fer86] Fertig, K., Andrews, A., and Wang, C., "Knowledge-Based Management and Control of Communications Networks", Milcom'86, 28.6.

[Goy87] Goyal, S., and Kopeikina, L., "Evolution of Intelligent Telecommunication Networks", Workshop on Expert Systems and Network Operation, IEEE, ICC, 1987.

[Had86] Haddawy, P., "Implementation of and Experiments with a Variable Precision Logic Inference System", AAAI, 1986.

[JPL86] "Technical Requirements for the GCF Upgrade Task", JPL, 4800 Oak Grove Dr., Pasadena, CA 91109, 1986.

[Kat87] Katsuyama, Y., "Expert System for Digital Transmission Network Trouble-Shooting", Workshop on Expert Systems and Network Operation, IEEE, ICC, 1987.

[Laf88] Laffey, T., Cox, P., Schmidt, J., Kao, S., and Read, J., "Real-Time Knowledge-Based Systems", AI Magazine, Spring 1988, pp 28-45.

[Lei87] Leinweber, D., "Real-Time Expert Systems for Space Applications", AFCEA Symposium "Space Technological Challenges for the Future", US Naval Academy, 1987.

[Mic86] Michalski, R, and Winston, P., "Variable Precision Logic", Artificial Intelligence, Vol 29, No 2, 1986.

[Moo86] Moore, R., "Expert Systems in On-Line Process Control", Proc. CPC-3 Conf., 1986.

[Pat85] Paterson, A., Sachs, P., and Turner, M., "ESCORT: the Application of Causal Knowledge to Real-Time Process Control", Expert Systems 85, Cambridge University Press 1985.

[Red86] Reddy, Y., and Uppuluri, S., "Intelligent Systems Technology in Network Operations Management", IEEE ICC, 1986, 39.1.

[Sau83] Sauers, R., and Walsh, R., "On the Requirements of Future Expert Systems", Proc. 8th IJCAI, 1983.

[Ter87] Terplan, K., "Communication Network Management", Prentice-Hall, Inc., 1987.

[Yeh87] Yeh, S., Ellis, C., Ege, A., and Korth, H., "Mean Value Performance Analysis of Two Locking Mechanisms in Centralized Design Environment", International Journal of Information Science, 1988.

N89 - 15553

Functional Reasoning in Diagnostic Problem Solving*

Jon Sticklen [1]
AI/KBS Group - CPS Dept
A714 Wells Hall - Michigan State University
East Lansing, MI 48824-1027

W. E. Bond and D. C. St. Clair [2]
McDonnell Douglas
Research Laboratories
St. Louis, MO 63166

1 Abstract

This work is one facet of an integrated approach to diagnostic problem solving for aircraft and space systems currently under development. The authors are applying a method of modeling and reasoning about deep knowledge based on a *functional* viewpoint. The approach recognizes a level of device understanding which is intermediate between a compiled level of typical Expert Systems, and a deep level at which large-scale device behavior is derived from known properties of device structure and component behavior. At this intermediate functional level, a device is modeled in three steps. First, a component decomposition of the device is defined. Second, the functionality of each device/subdevice is abstractly identified. Third, the state sequences which implement each function are specified. Given a functional representation and a set of initial conditions, the functional reasoner acts as a consequence finder. The output of the consequence finder can be utilized in diagnostic problem solving. The paper also discusses ways in which this functional approach may find application in the aerospace field.

2 Introduction

Over the last decade there has been a calling for deep-level reasoning capabilities [11,15] in knowledge-based systems. This calling has, in general, expressed two intuitions:

---

* Partial support for this research is from the McDonnell Douglas Independent Research and Development program. Dr. Sticklen is also currently supported as an Ameritech Fellow at MSU, and by equipment grants from Apple Computer and Texas Instruments.

[1] Part of the research reported here was performed while Dr. Sticklen was associated with the Laboratory for Artificial Intelligence Research, the Ohio State University.

[2] Dr. St. Clair is also Professor of Computer Science at the University of Missouri-Rolla Graduate Engineering Center in St. Louis, MO.

1. The first intuition is that domain experts have many layers of understanding about their areas of expertise, ranging from highly *compiled* knowledge to deep-level understanding which is not tuned specifically to a particular problem solving task. The ability to layer understanding has consequences both for problem solving and for the explanation of problem solving.

2. The second intuition is that approaches to knowledge-based systems which rely solely on patterns of data to establish working hypotheses will not, in the final analysis, be robust enough to support large systems. Although there have been convincing arguments that highly compiled systems can, in principle, be complete [7], there nonetheless is a practical difficulty in endowing a system with all of the patterns it is likely to need during problem solving. The central issue is that problem solvers should be able to deal with *novel* situations. If pre-stored patterns of data formed the only cornerstone to problem solving, it would be very difficult to deal with such novelty.

Although there is general agreement that deep reasoning capabilities are desirable, there is no apparent consensus on how to model such deep knowledge, nor on how to reason about it. In fact, there is no clear consensus on what the terms "deep-level understanding" or "deep-level reasoning" mean.

Recently, Sticklen [22,23] developed an integrated approach to diagnostic reasoning in the medical domain. One component of his diagnostic architecture is a deep-level reasoner which acts as a adjunct problem solver to a compiled level unit. The authors of this paper have recently begun to apply and extend Sticklen's approach in the domain of aerospace systems. In this paper, the rudiments of the methodology are described, its utility for aerospace problems is pointed out, and the points of contact between the research reported here and other lines of inquiry are described.

3 Functional Approach

Prior research in deep reasoning methods have typically fallen into two broad areas: the *causal net approach* [16,17,18,26] and the *naive physics approach* [1,9,10,12,13]. The causal net approach centers on representing deep knowledge via a causal net structure. Deep reasoning then means controlling the process of navigating the causal net. The naive physics approach centers on deriving the functionality of a device from the function and structure of its constituent subdevices.

On close examination, the causal net approach appears to be a compiled problem solving approach operating in the representation framework of a semantic net. The naive physics approach is aimed at *deriving* the behavior of a system from its components. However, there is a level of device understanding that is intermediate between compiled level understanding and the level at which composite behavior is derived. That middle ground of device understanding centers on the *known* purpose to which a device is put; i.e., on the function of the device and its subdevices.

Starting from the broad framework of the Generic Task (GT) theory of knowledge based systems [3,4,5], and building on the initial representational framework provided by Sembugamoorthy and Chandrasekaran [20], the present approach to representing and reasoning about devices *via* deep-level knowledge incorporates facets of the two other approaches.

3.1 Underlying Intuitions

Intuition 1: Limitation to Devices. First, concern is limited to the world of *devices*, where a device is defined as "a naturally occurring or artifactual assemblage whose purposes-goals-functions are known." Any physical object can, in principle, be described in terms of its physical properties, but once the *use* of a device is understood, comprehension has progressed to a higher level.

Device level understanding is distinguishable from the "attribute description" level by the indexing capability gained at the device level; e.g., if a car's purpose is understood to be centered around transportation, then its color attribute is irrelevant. The color attribute of a stealth aircraft, on the other hand, is highly relevant to its purpose, and hence would find a place in a functional representation.

Intuition 2: Device Decomposition. Device complexity is managed by *decomposing* the device into a number of components until a level is reached at which one can grasp how subdevices at that level operate. Understanding the operation of the overall device can then be stated in terms of the operation of the components.

Intuition 3: Indexing. As device decomposition proceeds, understanding of the device is organized in terms of the purposes of the device; i.e., in terms of its *functions*. It is important to note that in the functional approach, device functionality is given, **not** derived.

Intuition 4: Composability. Intuition #2 above dealt with decomposing a complex device into a number of subdevices. Intuition #3 is that we organize our understanding about each of those subdevices in terms of their individual functions/goals. Both #2 and #3 can be characterized as dealing with the *static* representation of device understanding. Note, that the complexity of device understanding is managed by a process of decomposition. Thus, the process of reasoning about a particular device in a particular situation must have the ability to dynamically select those parts of device functionality which are relevant to the current situation.

Intuition 5: Information Processing Task. The last intuition concerns the input-output relation of the task set for using the functional representation. In more formal terms, this is called the Information Processing Task (IPT) [14]. The IPT for functional level reasoning is a focused *consequence finding* in which the problem solver is given a set of initial conditions about either the state of the device or the nonavailability of some of the normal functions of the device. The output is the consequent changes that take place in state variables of the device.

3.2 Functional Representation and Reasoning

Building on earlier work by Sembugamoorthy and Chandrasekaran [20], Sticklen [22,23], developed methods of representing and reasoning about functional level phenomena, including a family of languages for representation and a full consequence finding algorithm. A description of this functional approach follows.

3.2.1 Representation

The functional level description of a device includes three core components: a description of device structure, a description of the functions of the device, and a description of the behaviors of the device which carry out given functions. To illustrate the various components of the functional representation, consider the simple device shown in Figure 1 and the partially complete functional description shown in Figures 2 and 3.

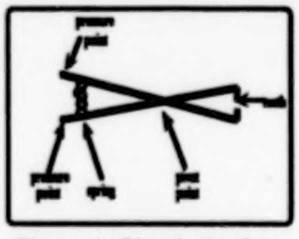


Figure 1: Simple Device - A Clothespin

Function: ***Open*** of ***CLOTHESPIN***
ToMake: teeth-more-open
Provided: (pressure-point-force-applied GreaterThan restoring-force)
By: ***OpenBehavior***

Figure 3: Open Function of Device Clothespin

Device: ***CLOTHESPIN***
SubDevices: (***ARM***,
PIVOT,
SPRING, ...)
Functions: (***Open***,
Close,
Hold,...)

Figure 2: Part of Device Specification for Clothespin

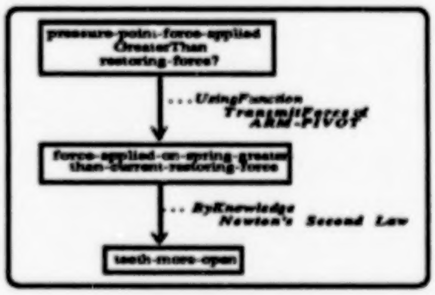

pressure-point-force-applied GreaterThan restoring-force?
...UsingFunction TransmitForce of ARM-PIVOT
force-applied-on-spring-greater than-current-restoring-force
...ByKnowledge Newton's Second Law
teeth-more-open

Figure 4: A Behavior of Device Clothspin

The device description shown in Figure 2 includes two items: a listing of the subdevices which compose the current device, and a listing of the functionalities of which the device is capable. It is possible to conceive of a clothespin as made of many different sets of subdevices. A characteristic of the functional level description is that there are many possible ways of decomposing a device into its components. The decomposition selected depends on the reasoning purposes for which the representation is constructed.

The function description of Figure 3 contains three items:

1. a *Provided* clause which states the conditions under which the function is applicable. The Provided clause may be thought of as a precondition for the behaviors which carry out the function.
2. a *ToMake* clause which states the result(s) that will be obtained if the invoked function completes successfully. The ToMake clause may be thought of as a postcondition for the behaviors that carry out the function.
3. a *By* clause which states the behavior(s) carrying out the function.

The function declarations within the functional description provide a means of knowing what can be achieved (the ToMake clause), what must be true for a given function to be applicable (the Provided clause), and a pointer for where to look if a more detailed description is needed of how the function achieves its goal (the By clause).

The description of behavior at the functional level contains two ingredients, as illustrated in Figure 4.

1. Partial state descriptions of the device. For example, *teeth-more-open* is a *partial* state of the Clothespin device. It does not represent a total state description of the Clothespin.
2. Annotated links between states. These links represent both the temporal flow of one state leading to another, and the *reason* that one state follows another. Note that the annotation could be a pointer to another function or behavior, or to a non-decomposable fragment of world knowledge.

From the above discussion, the following facets of the functional description of a device can be seen.

1. The functional representation of a device is a conceptual abstraction of what a device is and how it works. The "what it is" part is represented as a collection of subdevices related by a ComponentOf relation. The "how it works" is represented as the functionalities of which it is capable and the behaviors that accomplish those functions.
2. The functional description allows a natural modularity in understanding devices. This modularity is important in three specific ways. First, a subdevice of the overall device may be replaced with another totally different subdevice which accomplishes the same functions. Second, in understanding from the top level how the device functions, one is normally led via a chain of

   device => function => behavior => sub-device ...

   to lower and lower levels of sub-devices. However, this path of understanding may be terminated before the lowest levels of the device are reached. Once a level is reached at which a particular functionality of some underlying sub-device may be "assumed true," further probing along the current path is unnecessary. This ability to probe only as far as needed follows directly from the modularity of representation adopted.

Third, and most importantly, since the functional representation is organized around *fragments* of device understanding, the ability to dynamically compose answers is pivotal. Without significant modularity, such composability would not be possible.

3.2.2 Functional Reasoning

As noted above, the Information Processing Task of the functional approach is a type of consequence finding. The consequence finding is undertaken in response to a *particular set of conditions*, and amounts to building up a full state change diagram from the relevant fragments that exist in the behaviors of the functional representation. Sticklen [22] describes the operation of the consequence finding portions of the functional reasoner in full detail. Informally, the consequence finding steps in the functional approach can be described as follows.

First, the current nondefault conditions of the device are input. The consequence finder uses these initial conditions to *index* the applicable functions of the device by examination of the preconditions of each function. Through a filtering operation, the highest level applicable functions are selected as starting points. As one of the selected functions is taken, each implementing behavior is retrieved. A behavior is represented as a directed graph structure having two types of nodes: one representing changes in device state, the other representing a "knowledge pointer" to the reason for the state change. The knowledge pointers are of two types: decomposable and nondecomposable.

Next, the consequence finder recursively expands each one of the decomposable knowledge pointers. Not all functions and behaviors are applicable in a particular situation since all preconditions must be met before a function or behavior can be used. With that selectivity taken into account, the process is not unlike the process of *macro expansion* in typical computer science terms.

Following the second step, a particularized causal net like structure for the current device and situation is produced. Each node in this structure represents a change in a state variable in the device. The overall consequences on the device are then determined by simply traversing the structure and "adding up the results" of all the state changes that have occurred. Moreover, because a record of the preconditions of applied functions/behaviors can be easily maintained, it is also possible to state the consequence finding results in terms of *assumptions* which must hold in order for the derived consequences to be valid.

4 Functional Reasoning in Diagnosis

For diagnostic systems based on compiled classificatory problem solvers [2,6,21,22,24], each diagnostic hypothesis is represented by a classification specialist.

Each of these specialists must be able to establish its own viability, given the particular device/situation under consideration. One way of establishing this hypothesis is to rely on compiled associations in one way or another. To depend solely on such compiled knowledge assumes that novel situations will not be encountered. In general, this assumption is not valid.

The Function Level Consequence Finding strategy, as outlined above, provides a way to *derive* associational links that can be used for hypothesis establishment, and the assumptions that must hold for the links to be valid. In order for a compiled level diagnostic hypothesis to make use of a functional level consequence finder, the diagnostic hypothesis needs one additional knowledge structure - a representation of what the diagnostic hypothesis means in functional terms. Given such initial conditions, the functional reasoning process derives the consequences for the device. At the diagnostic level, *results* of the functionally based consequence finding can be used as patterns which should be present *if* a particular diagnostic hypothesis is valid for the current case. Sticklen has described this interaction between a deep problem solver and a compiled level problem solver in detail [22].

5 Functional Reasoning in Aerospace Applications

To date, the functional approach described has been applied primarily in the medical domain. However, a preliminary examination of several typical aerospace systems (electronic, hydraulic, and a fuel system) indicates that this approach is also applicable to engineered systems, especially as a aid to the design/redesign process, as well as in trouble shooting situations.

During the design stage, an engineer will initially define the high-level functionality of a device. Then the device design is refined by specifying subdevice functionality. When sufficient detail has been developed, specific components and interconnections are chosen to implement the low-level functionality. This process is fully supported by the functional approach described. In addition, the functional approach appears to support specific implementation requirements:

1. The ability to handle systems that exhibit both steady-state and dynamic behaviors (example-both the operation of a fuel system under constant demand conditions as well as the conditions that occur when the engine being supplied goes from 50% to 100% power). Many diagnostic procedures assume a system that is operating under steady-state conditions. Although many failure conditions can be detected using that assumption, other failures occur only during device transients.

2. The ability to identify situations that produce both "hard" failures and also situations where a device fails to perform "to specifications." Many diagnostic systems assume that a failure will occur in a distinct manner. This assumption is incorrect in some instances. Instead, the system may

malfunction by failing to perform to desired specifications or by producing symptoms which are not distinctive.

3. The ability to explicitly address device connectivity. Aerospace systems are often constructed by assembling a large number of interconnected components. The representation of connectivity should be explicit. This representation allows the designer to quickly wire together functional units. It also provides the ability to perform quick reconfigurations for design modifications.

To be successful, the approach must be able to interface with design and test environments. It is anticipated that the functional approach will allow considerations of testability to enter in the design process much earlier.

6 Discussion

In this report, a functional approach to representing and reasoning with deep knowledge of devices has been described, and a method to use the results of functional reasoning in diagnostic problem solving has been sketched. Recently, several other lines of research have been reported which attempt diagnostic problem solving from first principles, for example, see Reiter [19]. This recent body of work is largely an extension of the diagnostic outlook taken by Davis [8] and others who set out to perform diagnostic problem solving directly from a deep level. On the other hand, the functional approach described here integrates both a compiled level and a deep-level problem solver into a single composite unit in which the compiled level problem solver can play the important role of focusing the deep-level problem solver.

Beyond its uses in diagnostic problem solving for aerospace systems, a functional approach can also be applied to some of the problems associated with the issue of design knowledge capture [25]. Design knowledge capture is the process of organizing design knowledge in a machine-interpretable form. Design knowledge is composed of the specific design solution description together with the underlying rationale for the solution. The design description defines what the solution does and why it does so. The rationale includes the information used by the designers, the analyses that were performed, and the decisions that were made to produce the solution. Because the central aspect of the functional approach is the organization of knowledge about how a device works, it directly provides a description of what the device does and how it performs its function. It also provides an active representation which can be used in the future to simulate the operation of the device or to perform diagnostics.

7 References

1. Bylander, T. 1986. "Consolidation: A Method for Reasoning about the Behavior of Devices." Ph.D. The Ohio State University, Columbus, Ohio.

2. Chandrasekaran, B. 1982. "Decomposition of Domain Knowledge into Knowledge Sources: The MDX Approach." *Proc. 4th Nat. Conf. Canadian Society for Computational Studies of Intelligence.*

3. Chandrasekaran, B. 1983. "Towards a Taxonomy of Problem-Solving Types." *AI Magazine*, 9-17.

4. Chandrasekaran, B. 1985. "Generic Tasks in Knowledge-Based Reasoning: Characterizing and Designing Expert Systems at the "Right" Level of Abstraction." *Proceedings of The IEEE Second Annual Conference on Artificial Intelligence Applications*, IEEE.

5. Chandrasekaran, B. 1986. "Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design." *IEEE Expert*, 23-30.

6. Chandrasekaran, B., F. Gomez, S. Mittal and J. W. Smith. 1979. "An Approach to Medical Diagnosis Based on Conceptual Structures." *Proceedings of IJCAI 6.*

7. Chandrasekaran, B. and S. Mittal. 1983. "On Deep versus Compiled Knowledge Approaches to Medical Diagnosis." *International Journal of Man-Machine Studies*, 425-436.

8. Davis, R. 1984. "Diagnostic Reasoning Based on Structure and Behavior." *Artificial Intelligence*, 347-410.

9. de Kleer, J. and J. S. Brown. 1984. "A Qualitative Physics Based on Confluences." *Artificial Intelligence*, 7-83.

10. Forbus, K. D. 1984. "Qualitative Process Theory." *Artificial Intelligence*, 85-168.

11. Hart, P. 1982. "Directions for AI in the 80s." *SIGART News Letter.*

12. Kuipers, B. 1984. "Commonsense Reasoning about Causality: Deriving Behavior from Structure." *Artificial Intelligence*, 169-203.

13. Kuipers, B. and J. P. Kassirer. 1984. "Causal Reasoning in Medicine: Analysis of a Protocol." *Cognitive Science*, 363-385.

14. Marr, D. 1982. "Vision." W.H. Freeman.

15. Michie, D. 1982. "High-road and Low-road Programs." *AI Magazine*, 21-22.

16. Patil, R. S. 1981. "Causal Representation of Patient Illness For Electrolyte And Acid-Base Diagnosis." Ph.D. M.I.T., Cambridge, MA.

17. Pople, H. 1977. "The Formation of Composite Hypotheses in Diagnostic Problem Solving: An Exercise in Synthetic Reasoning." *Proceedings of IJCAI 5*, 1030-1037.

18. Pople, H. 1979. "Heuristic Methods for Imposing Structure On Ill Structured Problems: The Structuring of Medical Diagnosis." In *Artificial Intelligence In Medicine*, P. Szolovits. 119-189. Westview Press.

19. Reiter, R. 1987. "A Theory of Diagnosis from First Principles." *Artificial Intelligence*, 57-95.

20. Sembugamoorthy, V. and B. Chandrasekaran. "Functional Representation of Devices and Compilation of Diagnostic Problem-Solving Systems." In *Experience, Memory, and Learning*, J. Kolodner and C. Reisbeck. Lawrence Erlbaum Associates.

21. Smith, J. W. 1986. "RED: A Classificatory and Abductive Expert System." Ph.D. The Ohio State University, Columbus, Ohio.

22. Sticklen, J. 1987. "MDX2: An Integrated Medical Diagnostic System." Ph.D. The Ohio State University, Columbus, Ohio.

23. Sticklen, J. and B. Chandrasekaran. 1985. "Use Of Deep Level Reasoning in Medical Diagnosis." *Proceedings of The Expert Systems in Government Symposium.*

24. Sticklen, J., B. Chandrasekaran and J. R. Josephson. 1985. "Control Issues in Classificatory Diagnosis." *Proceedings of IJCAI 9.*

25. Sticklen, J. and B. Chandrasekaran. 1988. "Using a Functional Approach for Knowledge Acquisition." *Proceeding of 4th Banff Conference on Knowledge Acquisition*, in press.

26. Weiss, S., C. Kulikowski and A. Safir. 1977. "A Model-Based Consultation System for the Long-Term Management of Glaucoma." *Proceedings of IJCAI 5*, 826-832.

N89-15554

The Elements of Design Knowledge Capture

Dr. Michael S. Freeman
National Aeronautics and Space Administration
Systems Engineering Division
Systems Analysis and Integration Laboratory
Marshall Space Flight Center AL 35812

Abstract

This paper will present the basic constituents of a design knowledge capture effort. This will include a discussion of the types of knowledge to be captured in such an effort and the difference between design knowledge capture and more traditional knowledge base construction. These differences include both knowledge base structure and knowledge acquisition approach. The motivation for establishing a design knowledge capture effort as an integral part of major NASA programs will be outlined, along with the current NASA position on that subject. Finally the approach taken in design knowledge capture for Space Station will be contrasted with that used in the HSTDEK project.

Introduction

Although Design Knowledge Capture (DKC) has become a subject of strong interest in the Artificial Intelligence (AI) community, and is a critical element of some major new NASA programs such as Space Station Freedom, there still exists considerable confusion as to its nature, scope and feasibility. This is aggravated by an ambiguity inherent in the term itself, and the natural inclination to view DKC as just a "scaled-up" knowledge based system development task. In this paper I will try to clarify some of the differences between DKC and typical knowledge engineering projects, identify some of the challenges inherent in these differences, describe the approaches being taken in NASA to meet these challenges, and discuss the needs/benefits which justify the effort being made to develop a methodology for DKC. As will be discussed in more detail below, it is my opinion that an ability to integrate DKC with the traditional engineering activities which comprise a NASA system development project will be essential to the success of the missions we are undertaking in the next decade, and beyond. It is also an outstanding opportunity for NASA to develop new technology which can have a pervasive and significant impact on American industry, fundamentally changing the way we perceive technological progress. NASA is responding to this challenge by setting itself the goal becoming a world leader in the area of intelligent autonomous systems for aerospace applications, and has set up an aggressive program for research, development, validation, and demonstration of DKC technology. The Hubble Space Telescope Design/Engineering Knowledgebase (HSTDEK) Project is the primary focus for DKC development in NASA at the present time, although other significant activities exist. As the lead center for the HSTDEK Project, Marshall Space Flight Center has made a strong commitment to the utilization of knowledge based systems in future missions.

The Nature of Design Knowledge Capture

In order to distinguish DKC as a special type of knowledge engineering activity, it will benefit us to first review some basic nomenclature. Artificial Intelligence is "the part of computer science concerned with designing intelligent computer systems, that is, systems that exhibit the the characteristics we associate with intelligence in human behavior - understanding language, learning, reasoning, solving problems, and so on". [1] It is in the area of problem solving that AI has seen its greatest practical successes, in the form of "Expert Systems" which emulate in software and hardware the ability of some person (or persons) recognized as having exceptional skills in solving a particular type of problem. The ability of a computer system to mimic such skills can be very valuable, as has been widely discussed in AI, and business, literature of the past few years. The process of capturing such skills in a form ("Knowledge Representations") amenable to implementation in a computer system (a "Knowledge Base") is called "Knowledge Acquisition", and a number of highly sophisticated tools have been developed to support this type of activity. These range from symbolic languages such as LISP and PROLOG, to single-paradigm tools (or "shells") such as Rulemaster and VP-Expert, and on to multi-paradigm development environments such as KEE and ART. The ability of these tools to support development of expert systems and the commercial success of the systems built using those tools have reinforced each other, resulting in better tools for constructing expert systems. But it also has caused the development of these tools to focus closely on the needs of expert systems at the expense of more generalized capabilities. This has, in turn, constrained the pursuit of intelligent systems with broader objectives [2]. These broader focus intelligent systems are called "Knowledge Based Systems" (KBS), of which expert systems are a subset. Where expert systems attempt to capture the scarce expertise available only from one or a very few widely recognized experts, knowledge based systems attempt to formalize the more general problem solving techniques more typical of competent engineers. For this reason they are less tied to a narrow application area, or "domain".

The goal of a DKC effort, as stated for NASA's Hubble Space Telescope Design/Engineering Knowledgebase Project, is to enable major projects to capture the design/engineering expertise they have acquired during the development of their systems in a knowledge base capable of supporting multiple applications [3]. Because of the nature of the expertise to be captured, these applications are knowledge based systems rather than expert systems. The knowledge utilized in developing a major aerospace system spans many technical disciplines, applied by a large number of engineers. In the past, some isolated areas of this expertise might be singled out as especially valuable or unique, and an expert system development project initiated. The premise behind DKC is that, given the complexity, cost, and operational lifetime of current major systems, a much broader and better integrated cross section of the expertise will be required to insure long term mission success.

It is not possible at this point in the development of a DKC methodology to rigorously delimit the scope or type of expertise which must be captured. In order to approach the question at all, we must distinguish between a knowledge base and a knowledge based system. A knowledge base typically consists of a "domain model" of some activity or component of a system coupled with expertise on how the elements of that model interact in some context. A knowledge based system utilizes the contents of the knowledge base, augmented with application specific expertise, to achieve some goal. In expert systems the domain model is so specifically tailored to a specific narrow application that the domain knowledge is difficult or impossible to separate from the application. In other words, the contents of the knowledge base cannot easily be reused to support multiple applications. Since DKC is intended to result in a knowledge base capable of supporting multiple applications, this is a significant difference. One consequence of this difference is that tools developed to build expert systems provide limited support to a DKC effort. Another effect is that a thorough DKC effort can be expected to result in a knowledge base much larger than other knowledge engineering activities.

In addition to inherent large-scale, multi-application characteristics, a DKC knowledge base also must involve the integration of multiple technical disciplines. A typical expert system will focus on only one technical discipline, such as biochemistry, VLSI electronics, or mathematics. DKC, therefore, poses formidable systems engineering (as well as knowledge engineering) problems. A typical NASA development team may represent thirty or more major technical disciplines, with consultants in narrower areas. Between contractor and civil service engineers, there may be hundreds of people involved in the development process. Current technology is adequate to build isolated expert systems in most of these disciplines. The challenge for DKC is to develop a methodology which will integrate the expertise of these various disciplines into a knowledge base in which they can be jointly applied to solve problems which cross discipline boundaries. At present, a systems engineering approach can be facilitated in a DKC project by establishing from the beginning some target applications to be supported by the DKC knowledge base which involve multiple technical disciplines, and address the system at an integrated (as well as component) level. Two good candidates which currently require a great deal of manpower to support in NASA projects are fault diagnosis (including Failure Modes Effects Analysis and Critical Items Lists) and training for operational support. By insuring that the DKC knowledge base can support these specific functions, as well as meet its general knowledge capture goals, we can provide both a near term return on investment and enhanced long term support. The state of the art in knowledge engineering and/or project requirements may result in the choice of other applications. For example, support for evolutionary design might be a better application than training, but the use of a KBS to support general design is still in the realm of research.

Given the distinction made earlier knowledge bases and knowledge based systems, it becomes clear that the very term Design Knowledge Capture is ambiguous. Design is not a domain which can be modelled; instead, we have design expertise of differing levels of generality which can be utilized in various applications. We might therefore build a knowledge based *system* to support our design activities. But we also refer to the result of these design activities as the "design" of our system. In building a DKC knowledge base we certainly will want to capture as much of this design *data* as possible in our domain model to support many different applications such as fault diagnosis, operations planning, and even evolutionary design. Additionally, much of that expertise acquired during system development which we want to retain consists of knowing how to design systems such as this one, and why this one was designed as it was. With regard to DKC, it appears that the best way to resolve these ambiguities is to consider design data, and rationale behind specific design decisions, as elements of the domain model, and thus of the knowledge base. More general design expertise should be considered as components of a knowledge based system for design, even though it was applied in this specific design instance.

With these clarifications, the DKC goal stated earlier should be more easily understood. The nature of DKC makes it a tremendous, but not insurmountable, challenge to our fledgling knowledge engineering capabilities. Much work remains to be done to develop a workable methodology and adequate tool set for DKC. NASA has set itself the goal of being a leader in the development and application of this technology. There are a number of reasons why this technology is especially critical to NASA [4], several of which will be discused below.

Motivations for Developing a Methodology for DKC

As Plato recommended in his "Republic", concepts are often best understood by examining them in the context of systems where they are applied on a large scale. Many of the systems developed by NASA represent a vast effort by hundreds of people encompassing dozens of technical disciplines. By considering what motivates NASA to pursue the development of DKC methodology, it should become clear that many of these motivations also apply in smaller projects. This discussion will therefore focus on the rationale which has led NASA to commit itself to major DKC efforts in near-term, large scale projects such as Space Station Freedom.

The importance of intelligent or knowledge based systems for NASA missions has been explicitly recognized on several occasions, both within and without the agency. Perhaps the most striking was the enactment of Public Law 98-371, which stated a requirement that:

> "The Administrator shall establish an Advanced Technology Advisory Committee in conjunction with NASA's Space Station program and that the Committee shall prepare a report by April 1, 1985, identifying specific space station systems which advance automation and robotics technologies, not in use in existing spacecraft, and that the development of such systems shall be estimated to cost no less than 10 per centum of the total Space Station costs." [5]

The seriousness with which Congress approached the use of automation and intelligent systems is reflected in the goal established for NASA's System Autonomy Technology Program:

> "The general goal to establish and maintain NASA as a world leader in this area of intelligent autonomous systems for aerospace applications will be achieved by significantly advancing the required technologies, by validating these technologies in operational environments, and by developing and maintaining world-class technical expertise, facilities and tools within the NASA organization," [6]

The Systems Autonomy Technology Program (SATP) Plan has identified development of a system knowledge base as "the central, most important technology development area" and points out that "this process must start during the design phase, where the final design represents a first baseline of factual information from which factual knowledge for the system knowledge base can be extracted". [7]

Development of intelligent systems to support NASA missions has, so far, been a matter of building individual expert systems to support particular applications. This is a very expensive process for many reasons, including the scarcity of personnel trained in knowledge engineering. The most costly activity is that of knowledge acquisition; capture of the required expertise from several sources such as documentation, data products, interviews with design and test engineers, etc. There are, as mentioned earlier, a great number of applications which could be supported by knowledge based systems utilizing this expertise. Taking one example, consider the effort required to develop a fault diagnostic expert system for each of the major components for a system if we treat each technical discipline as requiring a separate KBS. There will be enormous duplication of effort if we build a electrical fault diagnostic expert system, thermal fault diagnostic expert system, mechanical fault diagnostic expert system, etc. Furthermore, the benefits of using an integrated analysis (where thermal data are used to help diagnose an electrical problem, for example) would be totally lost. When you next consider that there are many other applications which are desirable to support our missions that could also use much of the same expertise, such as command planning, power load scheduling, evolutionary design, and so forth, then it becomes very clear that an organized, consistent approach to knowledge acquisition is both more effective and more efficient. To maximize these benefits, knowledge acquisition should be done as part of a DKC effort integrated with the development process itself. It is simply not possible to maintain the "standing armies" of engineers now used to support short term missions such as shuttle flights or Spacelab missions in an era where fifteen to thirty year operational phases are planned. The recognition of this fact has led NASA to initiate DKC efforts on its major new programs.

Development of a Methodology for DKC in NASA

In 1987 NASA initiated the Hubble Space Telescope Design/Engineering Knowledgebase (HSTDEK) Project. The primary goal of the HSTDEK Project is to enable major NASA projects to capture the design/engineering expertise they have acquired during the development of their systems in a knowledge base capable of supporting multiple applications. In order to accomplish this, current knowledge engineering technology must be extended in several areas, the new technology must be validated, and a mechanism established for transferring it to users within NASA. Six specific objectives have been identified for the project:

1. Develop a methodology for constructing multi-application, large-scale knowledge bases.
2. Develop a methodology for acquiring knowledge from multiple domain experts representing different technical disciplines, and integrating it into a single knowledge base.
3. Develop an approach for integrating knowledge engineering into the traditional engineering activitiesof a system development effort.
4. Validate this new technology in the context of a major NASA program: construction of a deep, comprehensive knowledge base for the Hubble Space Telescope.
5. Develop an in-house knowledge engineering capability for NASA to apply this new technology and support its validation.
6. Establish a program for making this new technology available to major new NASA projects, beginning with Space Station and AXAF.

This is the major effort within NASA to develop and put in place a DKC methodology. It is a joint effort between Marshall Space Flight Center (MSFC) and Ames Research Center. The basic research described in the first objective is being pursued by the Knowledge Systems Laboratory at Stanford University. Knowledge engineers at Lockheed and MSFC are the primary researchers on the second objective, while the third objective is the focus of a grant with the University of Alabama in Huntsville. The last three objectives are being handled as MSFC internal activities. HSTDEK is a five year project, funded by the Office of Aeronautics and Space Technology at NASA Headquarters. As shown in the objectives listed above, it includes research elements as well as a demonstration of DKC technology in the Hubble Space Telescope domain, which is expected to be of direct benefit to the HST during Orbital Verification of the system, and also during long term operations.

There exists a high degree of similarity in the development process for most major NASA projects. It is expected that the methodology for design knowledge capture developed in HSTDEK and validated in the Hubble Space Telescope domain will be immediately applicable in support of Space Station, AXAF, and other major NASA projects. The long operational lifetimes of these projects require DKC to meet operational objectives. In addition to retaining the design and engineering expertise developed on these projects, which usually dissipates rapidly during their operational phase, a systems knowledge base incorporating this expertise will greatly facilitate the construction of knowledge based systems for multiple applications. The problem becomes that of enhancing the knowledge base with respect to that particular application rather than starting from scratch each time. The enhanced system autonomy and productivity of ground/flight crew will result in improved mission efficiency, an extension of our capability to achieve mission goals, and an improved probability of mission success. The projected benefits to the HST Program reflect a limited subset of the expected payoffs of this project, and will now be discussed.

The planned operational lifetime of the HST is fifteen years. Even if design data can be maintained over that period by current manual/electronic means, the design and engineering expertise which provides the context for that data cannot. The HSTDEK knowledge base will provide a vehicle for making that expertise available to HST personnel throughout its lifetime. As part of its technology validation effort, HSTDEK will produce two knowledge based systems which will support specific HST applications: HSTORE and GESST. The HST Operational Readiness Expert (HSTORE) will support the Orbital Verification mission at MSFC immediately following launch of the HST with regard to telemetry monitoring and fault diagnosis of the Electrical Power System and the Pointing Control System, as well as planning for a potential Maintenance and Refurbishment (M&R) mission. The use of this system to reduce the MSFC manpower required for one and a half years of limited operational support is being investigated. This is estimated to result in a 50% decrease in the contractor support required for this purpose (approximately five man years reduction), as well as a faster response to anomalies. The Ground-based Expert System for Space Telescope (GESST) will support HST operations at GSFC, adding support for the Data Management System, Instrumentation and Control System, Mechanisms and Structures, and Thermal Control System, as well as scheduling, training and design applications. The operational impact of GESST has not yet been evaluated, but should certainly reduce GSFC reliance on MSFC and contractor experts, expedite operational support, and possibly reduce manpower required for operations. A contractor developing an AXAF proposal has unofficially estimated that the staffing per shift for that program can be reduced from 10 to 3 by use of HSTDEK technology. Considering the backlog of astronomers wishing to use the HST, the use of GESST to quickly diagnose (or even forecast and prevent) failures could result in an effective increase in that most valuable commodity, observing time, by reducing down-time. Similarly, an enhanced capability to plan maintenance and refurbishment missions should result in their occurring less frequently, which is a cost-savings in both dollars, shuttle time, and HST observing time. Another payoff from HSTDEK will be the HST knowledge base itself, which will be made available to AXAF and other NASA programs as a design aid. In addition to the HSTDEK technology deliverables per se, ten MSFC engineers will be given the training and experience needed to develop knowledge based systems. As they leave HSTDEK to work in other projects, a significant payoff will be their enhanced ability to use this technology. Coupled with the exposure gained from use of knowledge based systems to support the HST, a very high visibility NASA program, this technology transfer mechanism should result in a new generation of sophisticated knowledge based systems in NASA. HSTDEK has already generated an increased awareness of the potential of knowledge based systems in MSFC management, and more interest in applying this technology to NASA domains in the AI community.

This increased awareness has directly resulted in the incorporation of a requirement for DKC in the Phase C/D Request for Proposal issued for AXAF. Both bidders on that contract addressed the DKC issue in their proposal, and the winner (TRW) will now proceed to implement their DKC program. It is inappropriate to discuss specifics at this point since negotiations a re still in progress, but the presence of even a limited DKC element in AXAF planning is a very positive step for the technology, and a validation of its perceived importance in NASA.

By far the greatest challenge, and the strongest requirement, for DKC in NASA is the development of Space Station Freedom. In response to the Congressional direction discussed above, NASA prepared a document titled Process Requirements for Design Knowledge Capture, whose objective is "to define design knowledge capture requirements placed on the work package contractors in support of the Space Station design community." [8] This document goes on to say that DKC is intended to include both design objects and designer's knowledge. The response of bidders for the work package contracts in the area of DKC varied greatly in both the depth and scope of their proposed approaches. The Space Station Program Office has given the responsibility for consolidating procedures and data relating to DKC to the Systems Engineering and Integration Information Planning Group. This group includes members from the Program Office, the Work Package Centers, and the International Partners. One current objective of the group is to revise the Process Requirements Document for DKC, and provide a coordinated input to the Space Station Program Requirements Document. Unfortunately, the planning for DKC at that level is now focused on use of the Technical Management Information System to capture text oriented design information, rather than integration of knowledge engineering tools with the traditional engineering activities. Discussions with the Space Station Strategic Plans and Programs Division as part of a recent Advanced Automation Study [9] has resulted in the definition of a three year task beginning in Fiscal Year 1989 to assess the ability of the Software Support Environment (SSE) workstations which will be used for development of all Space Station operational software to support the DKC activity. This task will

> "Build upon on-going Design Knowledge Capture (DKC) and use SSE Workstations as mechanism for performing DKC activities; define DKC capabilities (and information content requirements) by working with a fully designed spacecraft and assess the potential of the SSE Workstation and its software to perform portions of SS DKC; identify hardware/software modifications required to the SSE Workstation to support DKC."[10]

This task will apply and extend the experience gained in HSTDEK in order to develop a plan for initiating a full DKC effort for Space Station. This approach will treat knowledge based systems as elements of the planned SS operational software set.

NASA is therefore aggressively pursuing the development of DKC methodology and planning for its incorporation in major new programs, including the Hubble Space Telescope, Advanced X-Ray Astronomical Facility, and Space Station Freedom. Given the magnitude of the design and engineering efforts required to develop these systems, and their projected fifteen to thirty year operational lifetimes, there is simply no practical way to complete their missions without the use of Design Knowledge Capture.

References

1. Barr, A., and Feigenbaum, E. A. (Eds.) The Handbook of Artificial Intelligence, Volume One, William Kaufman, Inc., Los Gatos CA, 1981, Page 3.

2. Freeman, M. S., "An Investigation of Multi-User Expert Systems", Dissertation, University of Alabama in Huntsville, May 1987.

3. "Systems Autonomy Technology Program (SATP) Plan" (FY89 Draft), NASA/Ames Research Center, August 1988, page A-160.

4. Freeman, M. S., and Hooper, J. W., "Factors Affecting the Development of Expert Systems in NASA", First Conference on Artificial Intelligence for Space Applications, NASA/Marshall Space Flight Center and the University of Alabama in Huntsville, October 1985.

5. "National Aeronautics and Space Administration Research and Development, 98 Stat. 1227, Research and Program Management Report" Public Law 98-371, July 1984.

6. "Systems Autonomy Technology Program (SATP) Plan" (FY89 Draft), NASA/Ames Research Center, August 1988, Foreword.

7. "Systems Autonomy Technology Program (SATP) Plan", NASA TM 100999, NASA/Ames Research Center, December 1987, Page 16.

8. "Process Requirements for Design Knowledge Capture", JSC 30471, Johnson Space Center, December 15, 1986, Page 1-2.

9. "Space Station Advanced Automation Study Final report", Strategic Plans and Programs Division, Office of Space Station, NASA Headquarters, May 1988.

10. Program Operating Plan (POP) 88-2 Guidelines for the Space Station Transition Definition Program, Office of Space Station, NASA Headquarters, May 16, 1988, Page 2-11.

N89-15555

KAM : A TOOL TO SIMPLIFY THE KNOWLEDGE ACQUISITION PROCESS

Gary A. Gettig

Phase Linear Systems, Inc.
9300 Lee Highway
Fairfax, Virginia 22031

ABSTRACT

Analysts, knowledge engineers and information specialists are faced with increasing volumes of time-sensitive data in text form, either as free text or highly structured text records. Rapid access to the relevant data in these sources is essential. However, due to the volume and organization of the contents, and limitations of human memory and association, frequently a) important information is not located in time; b) reams of irrelevant data are searched; and c) interesting or critical associations are missed due to physical or temporal gaps involved in working with large files.

The Knowledge Acquisition Module (KAM) is a microcomputer-based expert system designed to assist knowledge engineers, analysts, and other specialists in extracting useful knowledge from large volumes of digitized text and text-based files. KAM formulates non-explicit, ambiguous, or vague relations, rules, and facts into a manageable and consistent formal code. A library of system rules or heuristics is maintained to control the extraction of rules, relations, assertions, and other patterns from the text. These heuristics can be added, deleted or customized by the user. The user can further control the extraction process with optional topic specifications. This allows the user to cluster extracts based on specified topics.

Because KAM formalizes diverse knowledge, it can be used by a variety of expert systems and automated reasoning applications. KAM can also perform important roles in computer-assisted training and skill development.

Current research efforts include the applicability of neural networks to aid in the extraction process and the conversion of these extracts into standard formats.

INTRODUCTION

As technology advances, reams of information are being put onto electronic media daily. The Federal government alone maintains thousands of databases and there are several agencies such as DTIC and NTIC whose primary mission is to provide information to user communities. With this wealth of information at hand, a tool to access and retrieve relevant information in a comprehensive and timely fashion would greatly enhance productivity. This is especially true when developing an expert system.

Knowledge acquisition is one of the biggest obstacles facing anyone who is attempting to build an expert system. More time is spent on it than any other phase of the expert system life cycle. The first step in knowledge acquisition is domain orientation. This initial gathering of information about a domain is where the Knowledge Acquisition Module (KAM) is the most powerful.

Knowledge engineers generally know little of the subject they are attempting to model. It is often necessary to spend weeks doing background research. This involves reading through many documents and databases, most of which are irrelevant.

During this process, the knowledge engineer may miss valuable information due to time constraints or lack of a structured methodology to locate it. The knowledge engineer may also miss important relationships between objects or ideas. This is caused by temporal relationships within the documentation such that the knowledge engineer can no longer remember previous data and how it relates to what is being read. In addition, spacial gaps in text of related information make it difficult to obtain a comprehensive understanding of a topic or follow lines of reasoning to a conclusion. The above problems are compounded if more than one person is assigned to do the background reading.

The Knowledge Acquisition Module is a word-based natural language processor designed to extract specific knowledge from large volumes of text based files. These files can be in the form of a wordprocessor document, ascii text file, or a highly structured database. Information can reside on disk, in memory, or be ported in via a communication line.

METHODS OF EXTRACTION

A variety of different methods are used to extract information from text. These methods can work at the word, syntax, semantic, or pragmatic level. KAM expounds on two of these methods. The first, context analysis, works at both the word and syntax levels. Words can have a myriad of different meanings when presented alone. Groups of words, however, can have a very specific meaning. This meaning defines an idea or context. For example, while scanning a paragraph the word "BAT" is found. "BAT" can mean several things when viewed by itself. "BAT" could be a noun or a verb, each having a variety of different meanings. Figure 1 gives a sample of some of the different meanings of the word "BAT".

Figure 1. Stand alone meanings of various words
Adapted from [Webster2]

BAT (noun) -

1. a solid stick
2. a sharp blow
3. a wooden instrument used for hitting a ball in various games
4. a paddle used in various games
5. rate of speed
6. a flying nocturnal mammal
7. a hag or a witch

BAT (verb) -

1. to strike or hit
2. to discuss at length
3. to wander aimlessly
4. to wink

BASE (noun) -

1. the bottom or lower part of something
2. a main ingredient
3. fundamental part of something
4. the starting place
5. bitter tasting compound
6. the four stations at the corner of a baseball field
7. a center of operations

BASE (verb) -

1. to make, form, or serve as a base for
2. to find a base or basis for (used with on or upon)

BASE (adj) -

1. of little height
2. low in place or position
3. resembling a villain
4. being of low value or inferior properties
5. lacking higher values

DIAMOND (noun) -

1. a native crystalline carbon (gem)
2. a square or rhombus shaped figure
3. something that resembles a diamond
4. a baseball infield

DIAMOND (verb) -

1. to adorn with diamonds

Figure 2. Conceptual Grouping of Battle
Adapted from [Websterl]

- Battle(noun) -
  - engagement
  - action
  - clash -
    - brawl, broil, conflict
  - assault -
    - aggression
      - offense -
        - aggression, assailment, onslaught
    - onfall, onslaught
  - attack -
    - charge
    - drive -
      - initiative, push
      - raid, blitz, militarization, seizure
    - combat -
      - action
  - contest -
    - competition, rivalry, warfare

- Battle(verb) -
  - war -
    - challenge
    - struggle -
      - endeavor, essay, attempt
    - scrimmage -
      - affray, skirmish, melee
    - assault -
      - engage, aggress, beset, storm
    - bombard -
      - blitz, bomb
      - shell -
        - bombard, cannonade, rake
      - barrage, strike
  - fight -
    - strive, debate, dispute, resist
      - buck -
        - dispute, duel, repel, traverse, pulverize, unseat, duel
  - oppugn -
    - contend

The next word that might be encountered is "BASE". "BASE" is even more complex than "BAT". As can be seen in Figure 1, "BASE" has different meanings as an adjective, a verb, or a noun. The third word located may be "DIAMOND", which has various meanings as well. But observe what happens when "BAT", "BASE", and "DIAMOND" are put together. The context that the paragraph was alluding to becomes apparent - that of baseball. This was accomplished without baseball being explicitly stated within the document. KAM allows the user to perform this type of analysis through a user defined topical grouping. This grouping can be done at the sentence level, through the use of rules, or at the paragraph level with special topical clusterings. Meanings can also be derived using KAM at the syntactic level by the order and spacial relationships of these words in the topical clusterings.

The second method, conceptual grouping, works in KAM primarily at the word level. It can, however, be modified to include phrases. Conceptual grouping is accomplished by linking with a thesaurus or dictionary of synonyms in order to fully cover the meaning of a specified topic. By specifying one topic and then calling a thesaurus, one will be able to discover links between related ideas that would have otherwise gone undetected by a person searching through large volumes of text. For example, suppose one wanted to discover how a certain document pertains to 'battle'. One could specify 'battle' in the topics list and have KAM help link to related words and concepts. One can even specify the conceptual depth one is required to achieve. Figure 2 shows how easy it is to achieve a high level of conceptual dependency.

KAM uses the thesaurus to break out the user specified topics into parts of speech. It then follows the same part of speech while digging deeper for concepts. For instance, the verb 'battle' chains to the verb 'fight' when looking for related concepts and not to the noun 'fight'. This is to curtail the combinatorial explosion which would result if no constraints were placed on the concept grouping. This can, however, be turned off if necessary. The part of speech of the word can also be specified when giving the topic.

As can be seen from Figure 2, going beyond the third processing level causes two problems to occur. First, the topics become circular and further depth traversal is unnecessary. This can be seen in the case when the noun 'battle' chains to 'assault', which in turn chains to 'offense'. 'Offense' then chains to 'aggression' which was already established by 'assault'. The deeper the level the more circular chaining becomes. The second problem, which is the most difficult to manage, is that a topic tends to veer too heavily from its intended course, resulting in unintended generalizations. The question of when to stop is a matter of how deeply one needs to go to understand or reveal the relationships that are trying to be uncovered. Conceptual grouping is also useful in picking up trends and biases in documents that would have otherwise gone unnoticed.

HOW KAM WORKS

KAM identifies word relationships by their contextual and conceptual dependencies. It will use any text-based or database file to formulate ambiguous or vague relationships, and non-explicit rules or facts into manageable clusters of related information. KAM can also integrate several documents to eliminate spacial gaps between relevant information. The driving force behind KAM is the heuristic file set up by the user. This heuristic file consists of rule forms used in the extraction process. These forms can have associated with them their own set of topics and exceptions. For example, Figure 3 gives a simple heuristic file set up to extract rule forms on the paragraph given below.

> "These instructions pertain to the successful care and operation of the MK-50 automatic weapons under humid, tropical climate conditions. Special precautions may be followed for properly maintaining the lubrication of the cartridge ejection mechanism. If moisture is allowed to build up inside the weapon, jamming may occur during automatic firing. The ammo clip may be removed if jamming does occur. Always be careful to set the safety before attempting to dislodge jammed shells from the barrel. Note that the MK-50 is quite unlike its Uzi counterpart in the fabrication of the bolt and firing pin mechanisms. Therefore, it is important to take precautions in the removal of the firing pin, lest the spring action will come loose."

The extracts produced from this example are presented in Figure 4. Several interesting points can be illustrated from this example. The first is the use of priorities to resolve conflicts that arise in the extraction process. The fact "important" was not extracted because of the higher priority of the rule "comma-lest". Second, the overall generality of the rule forms allows them to be applied generically to many situations. While this example is simple, large heuristic files can be developed and maintained for various applications. In addition, a general heuristic file can be used to extract information from a text file which can, in turn, be used to customize another heuristic file. This is useful when little is known about the particular domain. Thirdly, KAM can use these extractions to aid in constructing rules and facts for direct use by various expert system shells. Below is an example of how KAM incorporates two of the extracts into Goldworks frames.

```
(G-1 (PRINT-NAME ) (DOC-STRING KAM default frame) (IS
KAMFRAME) (SYNTAX RULE) (IF ( moisture is allowed to
build up  inside the weapon)) (THEN ( jamming may occur
during automatic firing)))

(G-2 (PRINT-NAME ) (DOC-STRING KAM default frame) (IS
KAMFRAME)  (SYNTAX  RULE)  (IF  (  jamming  does  occur))
(THEN ( The ammo  clip may be removed)))
```

This example does not imply that all extracts in the raw form are suited for direct placement into an expert system but a good percentage can be incorporated with careful design of the heuristic file.

Another feature of KAM is that it allows the user to perform context analysis and conceptual groupings on paragraphs so that related paragraphs are clustered together. This will eliminate spacial gaps in related information and allow special heuristic files to be designed specifically for these related paragraphs.

Figure 3. Sample Heuristic File

```
RULE if-comma
      IF [1+] , [1+] .
  FORM : IF 1 THEN 2
  PRIORITY : 1
  END

  RULE lone-if
      [1+] IF [1+] .
  FORM : IF 1 THEN 2.
  PRIORITY : 1
  END

  FACT important
      [1+] IMPORTANT [1+]
  FORM : NOTE 1 IMPORTANT 2
  PRIORITY : 3
  END

RULE comma-lest
     [1+] , [0-0] LEST [1+].
 FORM : IF NOT 1 THEN 3
 PRIORITY : 1
 END

 FACT always
      [1+] ALWAYS [1+] .
 FORM : ALWAYS 2
 PRIORITY : 3
 END
```

Figure 4. Rule Forms : Generated by KAM

1. IF moisture is allowed to build up inside the weapon
   THEN jamming may occur during automatic firing

2. IF jamming does occur
   THEN the ammo clip may be removed

3. IF NOT Therefore, it is important to take precautions
   in the removal of the firing pin
   THEN the spring action will come loose

4. FACT : Always be careful to set the safety before
   attempting to dislodge jammed shells from the
   barrel

RESEARCH TOPICS

There are several areas of research that are currently being addressed. The major thrust is in the area of resolving ambiguous pronouns within the rule forms. This is one of the more difficult problems to solve in natural language processing. The method currently being used by KAM is to allow a toggle between the source text and the extract generated so that the user can resolve any ambiguities. However, if communication lines are used, source reference is impossible. One way to address this problem is to have KAM aid the user in identifying pronoun references so that user would not have to go back to the original source. This aid would come in the form of KAM giving a selection of possible pronoun sources along with a certainty ranking. Current work is being done in this area with neural networks using back-propagation [Allen]. The idea is to have a neural network learn how to identify pronoun references by having it learn from previous examples and generalizing about new instances.

Other areas of interest include how to better control the level of conceptual grouping and how to more efficiently convert the extracts to standard formats such as schemas, frames, and user defined structures. The user defined structures hold the most promise since they allow the user to better control the extraction process. This would allow KAM to perform functions throughout the expert system life cycle.

CONCLUSIONS

KAM provides a simple, robust, and easy-to-use tool for knowledge acquisition. Through the use of context analysis and conceptual grouping, one can cluster paragraphs on selected topics. Specially designed heuristic files can then be used to extract rule forms from the clustered paragraphs. This is usually the most efficient way of approaching the problem of extracting particular information from huge quantities of text. This process can, however, be carried out in reverse.

KAM allows complete control over the extraction and clustering process. In addition to global topics and exceptions, each rule form can have its own special topics and exceptions. This allows the user to better refine the extraction process.

The user can specify to KAM the exact form to use during extraction. The user can also design a template that determines what form the extract will be in. This feature is useful when the extracts are going to be ported into an expert system shell or any other standard format.

The flexibility of KAM allows it to be tailored to just about any application. One of the more interesting ones is training and skill development in a particular domain. Knowledge and skills are obtained much faster when irrelevant information is filtered out.

REFERENCES

Allen, Robert B. "Natural Language and Back-Propagation: Demonstratives, Analogies, Pronoun Reference, and Translation", Proceedings of the First International Conference on Neural Networks, San Diego, June 21-24, IEEE Press, 1987.

[Webster1] "Webster's Collegiate Thesaurus", Merriam-Webster Inc., 1976.

[Webster2] "Webster's Ninth New Collegiate Dictionary", Merriam-Webster Inc., 1988.

N89-15556

Design of an Expert System for Estimating the Cost of New Knowledge in High Energy Astrophysics

Edward L. Bosworth, Jr., Ph.D.
Assistant Professor of Computer Science
The University of Alabama in Huntsville

and

A. J. Fennelly, Ph.D.
Chief Scientist, Applied Science Branch
Teledyne Brown Engineering
Huntsville, AL

The High Energy Astrophysics Costing Tool (HEACT) is a combined Expert System/Numerical Simulation to evaluate sensors and experiments proposed for space platforms in order to determine their expected performance and to assess the amount and quality of scientific information likely to be gathered by such experiments. The end product of this tool will be both a cost estimate for the experiment package as deployed and also an estimate of the amount of new scientific knowledge (both new data and reductions in the uncertainties of previous measurements) which would result from the actual deployment and operation of that experiment package.

The numerical simulation part of HEACT will contain components to 1) calculate the high energy signatures of both actual and proposed classes of astrophysical objects, and 2) model the response and resolution of the candidate sensors for observing those objects. The rule based part of the HEACT will be based on knowledge in the areas of experiment design, instrument selection, and system integration developed from experience with existing high energy astrophysics observatories, both orbiting and lofted by rockets or balloons.

This paper will present mainly results from the design of the rule-based part of HEACT. It will focus on the structure of the rules and the structures used to represent the knowledge related to the design and integration of such experiments. Methods for integrating the results from numerical simulations into the expert system will also be discussed.

PRECEDING PAGE BLANK NOT FILMED

N89-15557

Issues in Management of Artificial Intelligence Based Projects

P. A. Kiss
The BDM Corporation
950 Explorer Boulevard
Huntsville, AL 35806

Dr. Michael S. Freeman
Systems Engineering Division
Systems Analysis and
Integration Laboratory
Marshall Space Flight Center AL

1.0 Abstract

Now that AI is gaining acceptance, it is important to examine some of the obstacles that still stand in the way of its progress. Ironically, many of these obstacles are related to management and are aggravated by the very characteristics that make AI useful. The purpose of this paper is to heighten awareness of management issues in AI development and to focus attention on their resolution.

2.0 Introduction

For the purpose of this paper, the emphasis is on the subset of AI known as Knowledge Based Systems (KBS). However, much of the discussion can be extrapolated to other areas of Artificial Intelligence (AI) with minor modifications. In order to discuss the future needs of AI technology, it is useful to look at the present state of the art and some trends that have emerged. Over the past few years, a great abundance of small prototype KBSs have been built. These have been developed, using various Knowledge Engineering tools, to solve limited domain problems with reasonable success. However, larger systems that are tackling full scale customer problems have been much slower in coming. This has created a wait and see attitude toward the use of KBS in many customers minds.

Those organizations that are in the forefront of technology and have been developing KB systems, are looking more and more at integrating KBS into the mainstream of computing technology as opposed to stand-alone environments. Thus, an emphasis on total system approaches are emerging. System engineering approaches traditionally look at such things as: life cycle management, documentation, quality, testing, and cost among others. These trends lead to the conclusion that rigorous methodologies are needed for AI development and integration. It is the issues that arise from the desire to apply rigorous methodologies to AI that the rest of this paper is focused on.

3.0 Discussion

There are a number of perspectives of AI that can be examined. One is the view of the AI technologist. This view sees AI as a leading edge technology to be explored as a solution to every problem and developed as an art form. Then there is the view of the old style engineer that rejects AI as a bag of risky tricks. Perhaps the most important view is that of the end customers. They are looking for the solutions to problems at the best overall price. Since the customer is the person paying for the work, system engineers should have their needs in mind when designing and developing systems to solve problems. Let us examine the implications of this last perspective.

PRECEDING PAGE BLANK NOT FILMED

Although tremendous strides have been made by micro computers, most organizations use mini and main frame computers for the bulk of their computational problems. These environments typically include in excess of 100,000 lines of software (S/W) code, data bases along with their management systems running in an integrated fashion. Whether being added to or being designed and developed from the ground up, it is these environments that must be considered when looking at the use of AI to solve a significant customer problem. From this perspective, KBS is another piece of S/W that should run on (or with) existing/planned hardware and be integrated with the other functions of the environments. Accepting this premise, let us proceed by examining how KBS development might fit into the mainstream of S/W engineering and what the ramifications might be.

3.1 Typical Software Development Process. Perhaps the most rigorous S/W development methodology is the one developed for Department of Defense programs in the form of DOD Standard 2167A. Most other S/W development life cycles can be extracted as a variation or subset of the DOD one. Figure 3-1 shows a typical S/W development life cycle. Since it is familiar to most, only the briefest description of the phases is given here.

In the Concept Definition phase, the basic ideas for the systems are developed through studies and trade analyses. Here the requirements are developed and documented in the form of top-level system design specifications and operational concepts. These requirements may be allocated to major system components.

In the Preliminary Design phase, the Design Specifications is finalized and a preliminary design is generated. Here the generation of interface and data base specifications takes place along with the development and validation of critical methods (such as algorithms), and test planning. All of which is presented at a Preliminary Design Reviews (PDR).

In the Detailed Design phase, the system is finalized in terms of Detailed Specifications, interfaces, and data base specifications. System test plans are developed along with operations manuals, and prototype testing and simulation takes place. This phase culminates in a Critical Design Review (CDR). The Development phase consists of coding the software according to the designs and specifications. During this phase, detailed test procedures are also developed. The Formal Test phase consists of a hierarchy of test and validation activities. These incrementally test and integrate the system prior to final acceptance and delivery. In the Maintenance phase, the system is kept running properly with occasional corrections and updates as necessary.

3.2 Idealized KBS Development. Most KBS have been developed on a seat of the pants basis. The methodology applied was greatly dependent on the particular building tool being used and on the background of the developers. Presented below is a more formal and somewhat idealized KBS development cycle.

Being with a Problem Identification phase that analyzes the problems and determines which portions are applicable to a KBS solution. In this phase, basic concepts and approaches are developed for the appropriate domains along with a development plan. The key participants and their roles are identified and a cost and benefits analyses the effort is performed.

Next, the Prototype phase develops a full understanding of the domain and task via the building of an initial capability. This prototype is used to develop a detailed design along with performance criteria, test cases, and selection of the tools and target environment. During the Development phase, the prototype is expanded to its full functionality. The user interface is developed and it is converted to fit the target

Figure 3-1. Standard Software Development Life Cycle

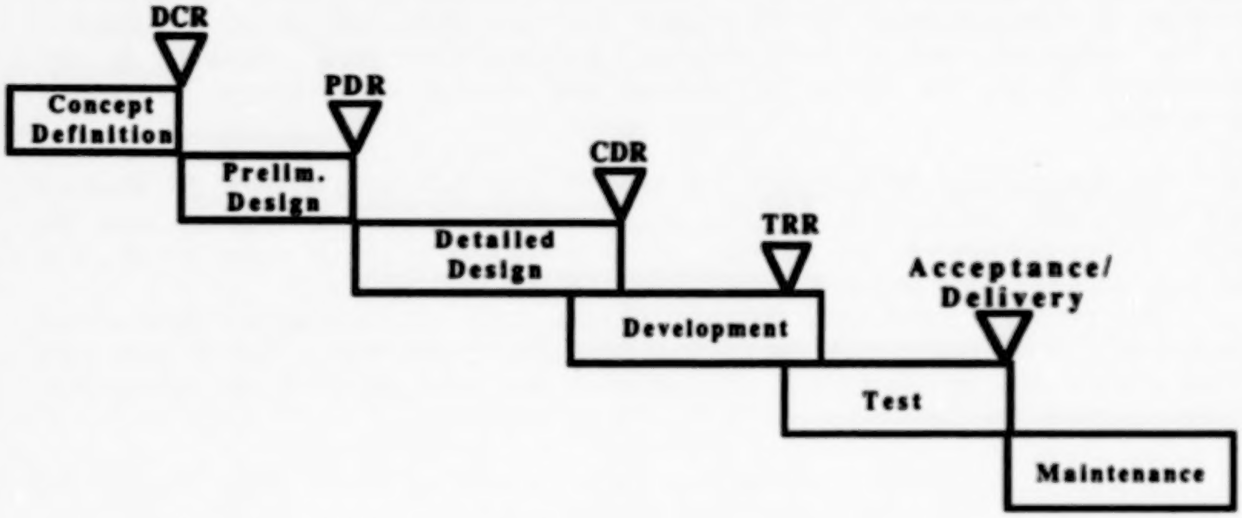
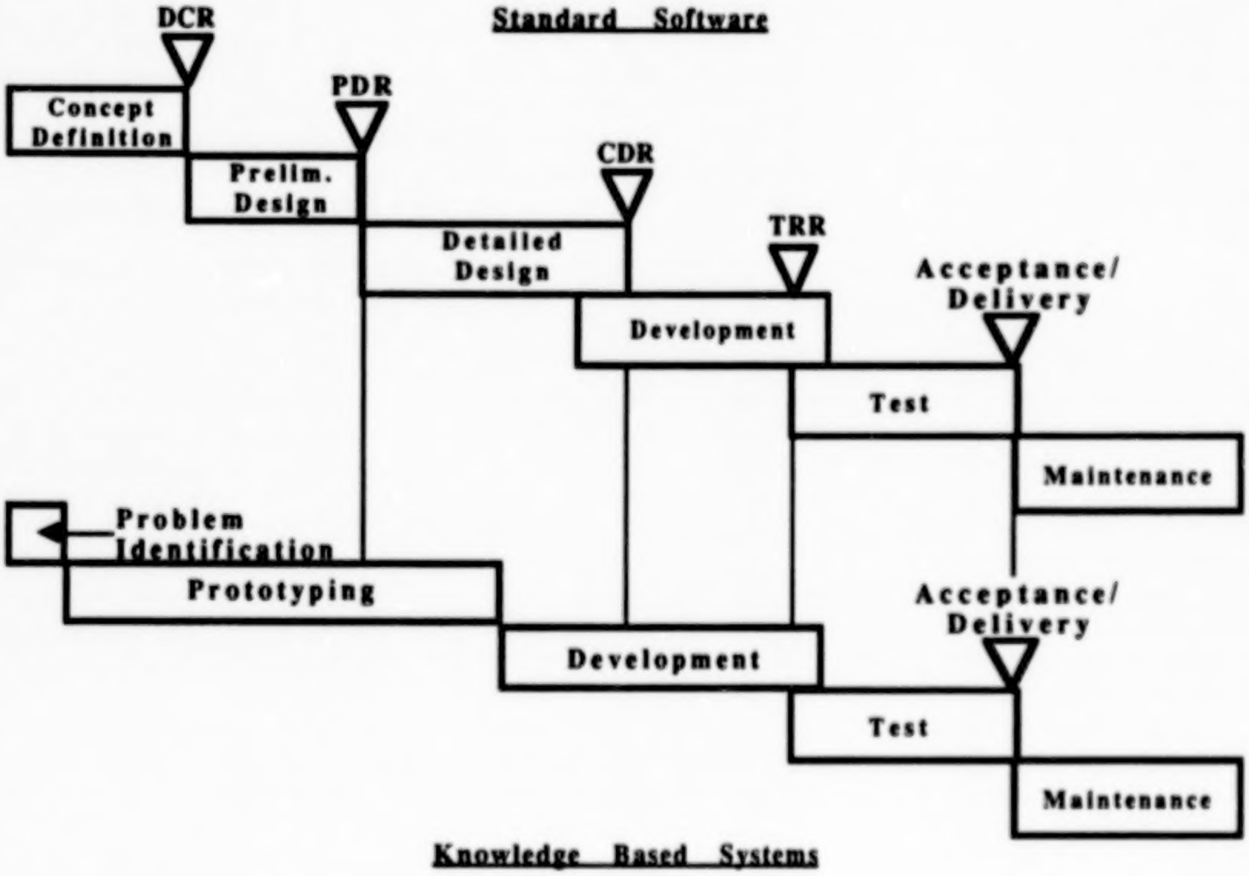


Figure 3-2. Integrated KBS Development in the Standard Software Development Life Cycle



environment. In the Evaluation phase, the system is tested against agreed upon criteria and is operated by experts against new scenarios. Here, if necessary, interfaces to other systems and data bases can take place and system performance can be enhanced before final delivery, documentation and training. In the Maintenance Phase, the system is corrected and updated as necessary for optimal operations.

3.3 An Integrative Methodology. In order to gain full benefits from the Idealized KBS methodology above, it needs to be integrated into a typical S/W development life cycle. Figure 3-2 shows how the two may be overlayed in an integrated development that contained a KBS component imbedded in a larger S/W system.

This proposed combination naturally imposes some of methodology rigor of the standard S/W development cycle onto that of the KBS components. That in turn leads to the examination and discussion of weaknesses that exist today in the management of Knowledge Based Systems.

3.4 Areas to Develop. By systematically examining each phase of a KBS development cycle, and comparing it to the corresponding standard S/W life cycle phase, areas needing development come to light. A top-level cut has been done, and below are some of the areas ripe for further attention and development.

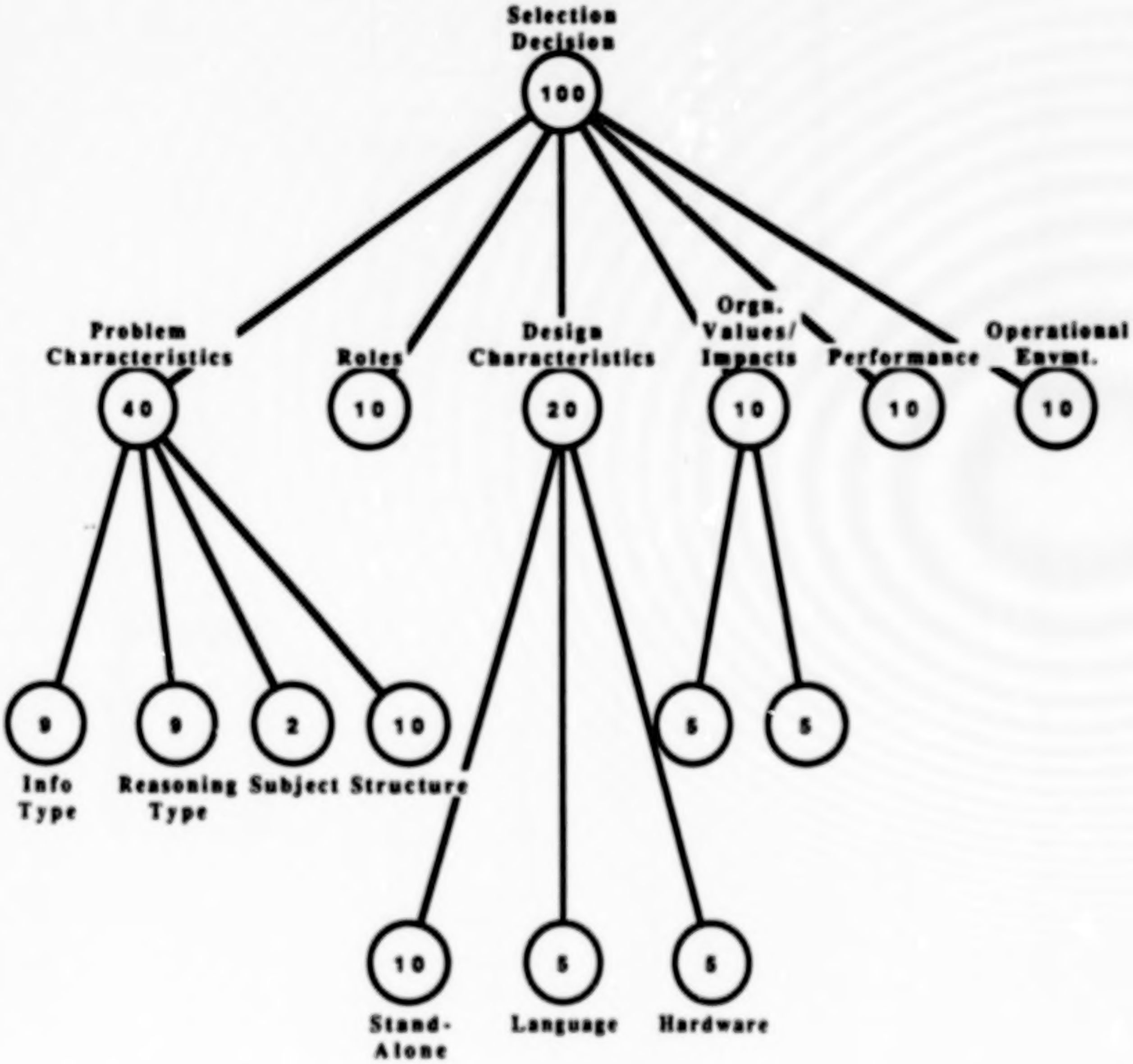
Beginning with the problem Identification phase (in Concept Definition), the first question should be "what are proper applications for KBS solutions?" This is not a trivial question since many problems can be solved by classical methods better than by Knowledge Based ones. Although many simple tests of applicability exist, there are few in depth methodologies widely accepted. It is suggested that such a methodology (exemplified in Figure 3-3) can be developed by assessing: problem characteristics, future role of the subsystem, design characteristics, organization values and impacts, required performance, and operational environments, to name the major areas. Decision trees can be put together by decomposing the above areas to lower levels and adding weights to them as appropriate. Once the methodology is fully developed, it can be calibrated by running it against existing KBS and their associated successes in the field.

In conjunction with the technology selection, process should be a cost/benefit analysis. This might be based on such things as the benefits of replicating an expert, or the savings compared to solving the problem via a different approach. More specifically, the key cost areas for developing a KBS are; knowledge engineering time, domain expert's time, users' time, design, development, and test for prototype and delivered KBS, hardware, and management. The benefits may include; increased productivity, new services or products, elimination of systems or procedures, improvements in quality, fewer or less qualified staff needs, and increase in equipment life. The above factors need to be quantified for each system and traded against each other.

For the sake of this discussion, let us consider a medical advisor application. Such applications are widely accepted as appropriate for a knowledge based system, as opposed to standard software. For this reason, we can expect that any cost/benefit assessment will be favorable.

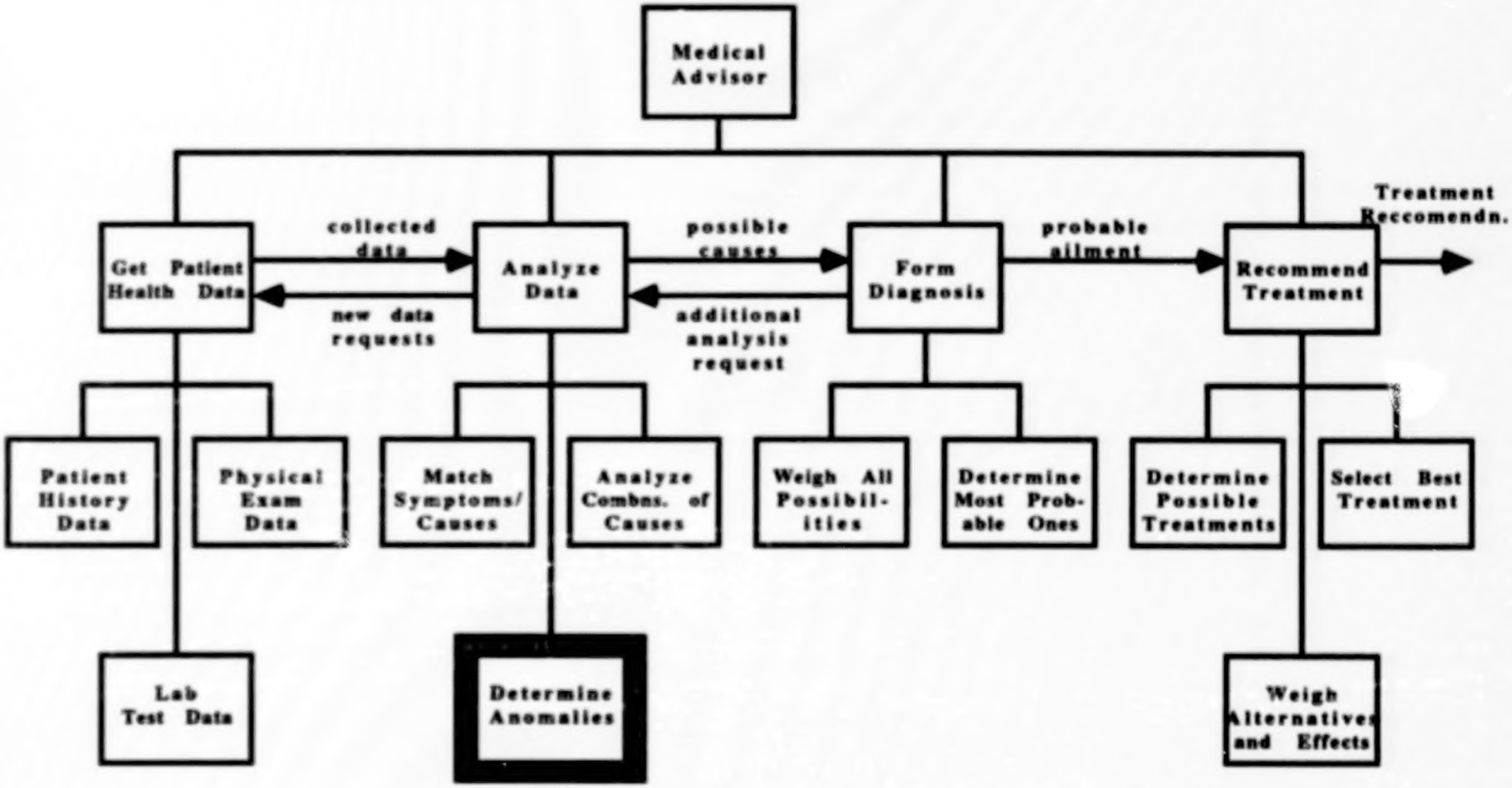
The next hurdle during Concept Definition (early prototyping) is to characterize the requirements/specifications. One method, shown in Figure 3-4, is to capture the functional requirements and decompose them as far as practical. Once a decomposition exists, the relationships between functions can be developed. This process is important because it will highlight the boundaries to the applicability of of KBS solutions. Taking the perspective that some parts of a problem can be straightforward and solvable by conventional methods may reduce the complexity of the KBS

Figure 3-3. Selection Methodology for KBS Applications



0-50	Non-KBS Application
40-80	Marginal
70-100	KBS Application

Figure 3-4. Sample Functional Decomposition for a KBS



components. One example is the "Determine Anomalies" function in Figure 3-4. This may be accomplished by a conventional database-driven limit checking program. Another example is the area of scheduling problems. Some solutions can be broken up into heuristic and algorithmic components. In addition to functional specifications, performance specifications should be developed for KBS. These are important since they affect the hardware and software selections made for the systems. The most important performance characteristics can be along the lines of quality, quantity, speed, or interfaces. In the medical advisor example, quality can be specified in terms of the percentage of correct diagnoses and/or recommendations. Quantity or speed can be assessed in terms of how quickly (once the required input data is provided) a diagnosis or recommendation can be completed. Finally, the interfaces should be specified in terms of the user's needs and the delivery environment. In this case a doctor may be the user, with an interactive interface as well as access to a database of patient history information, and traditional software to perform initial anomaly screening.

As the development of a system moves through the design phases, much emphasis is placed on documentation and reviews. In the case of KBS, documentation standards and review milestones are not well defined. The establishment of Preliminary and Critical Design Review (PDR, CDR) milestones for KBS should be considered an important part of system management. They are valuable for the developers to work toward, and for the customers/reviewers to understand the product being developed and gain confidence in its capability.

At PDR the knowledge gained from prototyping should be presented as it relates to the full KBS. The detailed functionality and performance requirements of the KBS should be documented. Some possibilities for the documentation are: functional decompositions, logic networks, decision flows, interface diagrams, and operational concept descriptions. Naturally, some of the KBS development tool outputs can be used for documentation, but most are usually more appropriate for a CDR level review. At CDR, the focus should be on the full integrated system. The knowledge base should be complete and captured by the development tool. Text level documentation should be provided describing interfaces between the KBS and other system components. Full screen level user interfaces should be defined and presented. There is a considerable amount of work still to be done in the area of KBS documentation standards.

A final area to be addressed here is that of testing and validation of KBS. This is perhaps their most important and also weakest area from the perspective of traditional operational systems. It is important because operational systems need to be trusted by users. Failure of a $300M dollar satellite, the life support system of a space vehicle, or your doctor's diagnostic advisor is likely to be viewed by users as not merely undesirable, but catastrophic. If a KBS is to be a key component of successful operation, then users will demand a high degree of confidence in their reliability. The AI community is of several opinions with regard to methods for testing KBS, ranging from the position that testing KBS is no different from testing any other software system, to a belief that KBS are so different that present methods cannot be applied to them. Nevertheless, major KBS have been built and put into operation successfully using variants of traditional; methods. Development of a reliable methodology for verification and validation of KBS must still be considered a high priority issue from the management perspective.

Some approaches can be outlined for achieving the required level of reliability in KBS. One is to explore the use of multiple concurrent KBS to provide redundancy in critical applications. This can be done through the use of multiple copies of the KBS in a voting arrangement, such as is used with traditional software systems in the shuttle today. Another is to develop multiple KBS addressing the same application, but drawing on different knowledge sources to cross validate each other.

A third approach is to develop the same KBS using different development tools. This latter approach is more cost effective than the second because knowledge acquisition (typically the most costly activity in KBS development) is done only once.

4.0 Conclusion

There are other areas of the software development life cycle which give rise to problems when applied to KBS. As the technology matures, and the issues discussed here are resolved, other issues will become more important. However, progress toward resolution of these issues, and development of a methodology for building KBS, will be crucial in gaining the support by management required for KBS technology to be integrated with traditional systems in operational environments.

N89-15558

Development of an Expert Planning System for OSSA

by

B. Groundwater, M.F. Lembeck, and L. Sarsfield

Science Applications International, Inc.
400 Virginia Avenue, S.W., Suite 810
Washington, D.C. 20024

Work performed under Contract #NASW-4092
Technical Monitor:

Mr. Alphonso Diaz
Assisttant Associate Administrator
NASA Office of Space Science andApplications

This paper presents concepts related to preliminary work for the development of an expert planning system for NASA's Office for Space Science and Applications (OSSA). The expert system will function as a planner's decision aid in preparing mission plans encompassing sets of proposed OSSA space science initiatives. These plans in turn will be checked against budgetary and technical constraints and tested for constraint violations. Appropriate advice will be generated by the system for making modifications to the plans to bring them in line with the constraints.

The OSSA Planning Expert System (OPES) has been designed to function as an integral part of the OSSA mission planning process. It will be able to suggest a "best plan," be able to accept and check a user-suggested strawman plan, and should provide a quick response to user requests and actions. OPES will be written in the "C" programming language and have a transparent user interface running under Windows 386 on a Compaq 386/20 machine.

The system's sorted knowledge and inference procedures will model the expertise of human planners familiar with the OSSA planning domain. Given mission priorities and budget guidelines, the system first sets the launch dates for each mission. It will check to make sure that planetary launch windows and precursor mission relationships are not violated. Additional levels of constraints will then be considered, checking such things as the availability of a suitable launch vehicle, total mission launch mass required vs. the identified launch mass capability, and the total power required by the payload at its destination vs. the actual power available. System output will be in the form of Gantt charts, spreadsheet hardcopy, and other presentation quality materials detailing the resulting OSSA mission plan.

N89-15559

PLANNING ACTIONS IN ROBOT AUTOMATED OPERATIONS

A. Das
Computer and Info. Sci. Dept.
Alabama A&M University
Normal, AL 35762

ABSTRACT

Action planning in robot automated operations requires intelligent task level programming. Invoking intelligence necessiates a typical blackboard based architecture, where, a plan is a vector between the start frame and the goal frame. This vector is composed of partially ordered bases. A partial ordering of bases presents good and bad sides in action planning. Partial ordering demands nonmonotonic reasoning via default reasoning. This demands the use of a temporal data base management system.

INTRODUCTION

Advanced technology for the space station and the US economy necessiates substantial use of general purpose automation and robotics requiring new generation machine intelligence and robotics technology. Three years ago, on this issue, NASA's Advanced Technolog Advisory Committee published a set of 13-point recommendations[1]. Intelligent plan adoption by robots is one major vital curriculum. Its ultimate purpose is to imply multi-level environment perception and modelling, decisional autonomy ranging from general planning to specific task operating, autonomous mobility capacity, sophisticated high level man-machine interface and efficient execution control systems. However, vagaries of the real world , its geometry, inexactness and noise pose large practical problems to the researcher and this forces investigations to have excercised on a handful of toy examples.

Recently, an attempt has been made to create an architecture for simulating intelligence in robot automated assembly operation[2]. In this architecture the reasoning system works with a two-dimensional system configuration, a task level configuration (embeded to high level plans and common sense reasoning) and a robot level configuration (numeric activities to live with ordinary geometric world). In programming robots the task level operations are specified according to their expected effects on objects, detailed kinetics of motion even as functions of inputs are not considered directly. In the process of task level programming every new robot level task(geometric world) is seen as a clusture of incremental planning at the task level(plans and reasons). Any problem in this incremental planning working with this two-dimensional system configuration is resolved by a blackboard based problem solver[6]. In this blackboard, all classes of temporal, spatial and event class

PRECEDING PAGE BLANK NOT FILMED

relationships invoked by the task structure are examined, and suitable prescriptions generated. The current architecture looks at task level plans vectors connecting the start frame to the goal frame and are composed of bases <R,T>s, where R represents robot level parameters and T represents task level plan configuration operative in the geometric world given by R. <R,T>s are seen to form a partially ordered set to admit mutations with time.

This paper gives a closer look at the preconditions existing in action planning for a robot at the task level. The implicability of partial ordering of <R,T>s is questioned and associated problems are formulated with common sense reasoning. It is observed that this leads to reasoning by default[9] bringing task level planning in the paradigm of nonmonotonic reasoning. A demand for a temporal data base system[3] for action planning seems inevitable.

PLAN VECTOR ON A BLACKBOARD

In ref.2 a plan is a vector in the robot level task level configuration space:

$$C_1\langle R_i,T_j\rangle + C_2\langle R_m,T_n\rangle + \ldots\ldots + C_n\langle R_y,T_z\rangle \qquad \text{(Eq.1)}$$

where $C_1, C_2, \ldots\ldots, C_n$ are real or complex numbers. <R,T>s are bases that mutate with time, also. They comprise a partially ordered set. A blackboard is a problem solver[3] enriched with highly domain specific heuristic knowledge. Ref.2 discusses the diagnosis of the simple case of the movement of the robot arm on a blackboard. Since the robot does not do only one single job, and since most of the tasks require repetition of the same subtasks several time, a comparative diagnostics is attributable on the black board. Two or more plans are compared side by side. Searching "plan-invoking macros" becomes more effective. There are problems, however.

WHY PARTIAL ORDER ?

In eq.1 <R,T>s are partially ordered bases. This gives freedom in forming the plan vector intelligently. The total geometric space configuration assumed in performing a job may then be seen as composed of a series of subtasks. Each of these subtasks was constructed out of realizable <R,T>s. Thus two plan vectors designed for totally two different jobs may be known as differeing by additions, subtractions and modifications of some <R,T>s. One plan, say, is goint to the grocery store and another, going to the doctor's office. These two are implemented in this way:

Grocery	Doctor's Office
Go 2 miles straight.	Go 2 miles straight.
Turn left, go straight.	Turn left, go straight.
Turn left, go straight.	Turn left, go straight.
Turn right, go straight.	Turn right, go straight.

Turn left, go straight.	Turn left, go straight.
Turn left, go straight.	Turn right, gostraight.

These two plans differ in their final turn. Bases <R,T>s are partially ordered in both these plans. Therefore, if one plan is implemented successfully and the other is not, a comparative diagnostic measures can be worked out(possibly searching common macros in both the cases). If on the other hand <R,T>s were totally ordered, the two plans differ without any flexibility of having a match between them.

POSSIBLE TROUBLES IN PARTIALLY ORDERED <R,T>s

The aforsaid example of comparing two plans on the basis of partially ordered bases <R,T>s requires that a strong table management system admonishing context dependent properties of <R,T>s need be present(A comparative diagnostics of a faulty plan, or, an working plan showing bugs later requires all the macros in correct order invoking the two plans). If in the second plan, for example, a fault is observed in the last right turn, then the comparative diagnostics rquires an account of the past history in correct order in both the plans. If you have known how to go to the grocery store(Plan 1) then going to the doctor's office (plan 2) needs a little modification in the final phase. Tracing the two plan vectors side by side with <R,T>s implementing them is necessary in case of bugs observed in one (or both) of them. Such tracing of history also requires cataloging and comparing time of occurrence of all subtasks, their duration needs to be noted too. To understand why you could not reach the doctor's office requires answering a question like how long did you take to perform the first two left turns, for example. Temporal ramifications of <R,T>s and managing their order of context dependency causes trouble in working with partially ordered <R,T>s. Obviously, it is a horrendous task.

THREE MORE PROBLEMS

Three more problems will arise in comparative diagnostics, infact, in any reasoning about action. These three problems are the frame problem of McCarthy and Hayes[8], qualification problem of McCarthy[7] and the ramification problem of Finger[4].

The frame problem enters into comparative diagnostics when two or more plan vectors are compared basis by basis, or subtasks by subtasks to determine which of them remain invariant in time while the action is taking place. If I succeed in going to the grocery shop but fail to go to the doctor's office, it was necessary to be determined that these two plans differed only in their final turn (as seen before). No interim unwarrented turn is admissible in both the plans, they were framed by preconditions.

This framing of preconditions lead to the second problem called qualification problem. It arises because the number of preconditions are always very large. Imagine all of the possibilities that prevented me in taking the last right turn while going to the doctor's office. Probably, my car broke down. probably, it was raining heavily and I missed the road sign. Probably, I stopped somewhere before the final turn to buy coffee and while taking off took a different direction. Probably, in the last right turn there was a detour sign forcing direction change. It is very unaffordable to pin point all these worldly possibilities.

The third problem is the ramification problem which is very severe in comparative diagnostics because it is unreasonable to explicitly record all the consequences of actions. In both our examples of plans on going to grocery and doctor's office a great number of possible consequences may occur which may not have any consequence at all. In going to the grocery store after making the first left turn I may see the fish market and buy some fish and after the next turn I may find my sister's home nearby and deliver part of the fish to her. The applicability of these ramifications for one plan will not be the same for the other. Moreover, the comparative diagnostician will not be able to work out which ramifications are supposed to show up any time for any plan vector under investigation. Inference in default logic may be a way out.

INFER $\langle R_x,T_y\rangle$ FROM THE INABILITY TO INFER $\langle R_m,T_n\rangle$

All the ramifications of any plan vector must be expressible using bonafide bases <R,T>s. It turns out that if there are n <R,T>s in a plan vector, any one single new <R,T> for admitting a new event or action in the plan vector will require n verfications for consistency with existing constraints. Therefore, reasoning with default logic automatically sets in: facts persists in the absence of information to the contrary. If I want to compare my faulty plan not leading to the doctor's office with the successful plan leading to the grocery store, all the possible ramifications that may be present in both successful and unsuccessful plans needs to be assumed existing, because they cannot be verified. This is expressed using Reiters default rules[9]:

$$\frac{\langle R,T\rangle_t : \langle R,T\rangle_{do(a,t)}}{\langle R,T\rangle_{do(a,t)}}, \qquad \text{(Eq.2)}$$

Which states that if <R,T> is true in time (or situation) t and <R,T> are still consistent after the action a, then we can infer <R,T> after the action. Computational problems still persists, though. To determine what is true after an action has been performed, the default frame axiom must be examined once for every fact of interest[5].

PLANNING ACTIONS

Within the context of above mentioned observations and conditions planning actions needs a strong nonmonotonic reasoning system for its support. The three immediately visible reasons for this are: the presence of incomplete information requires default reasoning, a changing world must be described by a changing data base, temporary assumptions about partial solution may be required for generating a complete solution[10]. In the present model the bases <R,T>s chronologically mutate in time between the start frame and the goal frame. This manifests a temporal data base system which is an extension of clasical predicate calculus data base. Mutations of <R,T>s also means that temporal information is incomplete, that is, our knowledge on the occurrence of events totally admits to partial ordering of <R,T>s to implement a plan. Such a system can be fruitfully dealt with a data base system called the time map manager or TMM[3]. Such a TMM admits shallow temporal reasoning to be consistent with the default reasoning system, because by default a deductive reasoning system works with a small number of calculations. Shallow reasoning systems provide the TMM with a mechanism for monitoring the continued validity of conditional predictions. Thus, the TMM rearranges tasks to take advantage of existing preconditions and warns of unexpected dangerous interaction between the effects of unrelated tasks.

CONCLUSION

The comparative diagnostics on a blackboard with the help of a time map manager is akin to visually scanning a massive amount of data organized in the form of a map. This brings a graphical picture to action planning. Default reasoning makes this action planning a nonmonotonic shallow reasoning system. Hints are there that using the time map manager a comparative diagnositcs on a blackboard may be able to overcome some classical problems associated with partial ordering of <R,T>s. It is currently under investigation.

REFERENCES

[1] Cohen, A. and Erickson, J.D., "Future Uses of Machine Intelligence and Robotics for the Space Station and Implications for the U.S. Economy", IEEE Journ. Robotics & Automation, Vol. RA-1, No.3 (1985) 117-123.

[2] Das. A. and Saha. H., "Embedding Intelligence In Robot Automated Assembly", Proc. First Int. Conf. on Indus. & Engng. Appln. of A.I & Expert Systems, Tullahoma, Tennessee, June 1-3 (1988) 1083-1088.

[3] Dean, T.L. and McDermott. D.V., "Temporal Data Base Management", Artif. Intell. Vol. 32, No-1 (1987) 1-55.

[4] Finger, J.J., "Exploiting Constraints In Design Synthesis", Ph.D. thesis, Stanford Univ., Stanford, CA (1987).

[5] Ginsberg, M.L. and Smith, D.E., "Reasoning About Actions I:

A Possible Worlds Approach", Artif. Intell. Vol. 35, No. 2 (1988) 165-195.

[6] Hayes-Roth. B., "A Blackboard Architecture For Control", Artif. Intell. Vol. 26, No. 3 (1985) 251-321.

[7] McCarthy. J., "Epistomological Problems of Artificial Intelligence", Proc. IJCAI-77, Cambridge, Mass (1977) 1038-1044.

[8] McCarthy.J. and Hayes.P.J., "Some Philosophical Problems From The Standpoint of A.I." in Machine Intelligence-4, ed. by Meltzer.B. and Michie. D., American Elseveir, N.Y. (1969) 463-502.

[9] Reiter. R., " A Logic For Default Reasoning", Artif. Intell. vol. 13, No. 1 (1980) 81-132.

[10] See for example, Artificial Intelligence by Rich. E., McGraw-Hill Book Co., NY, 1983.

N89-15560

Integration of Task Level Planning and Diagnosis for an Intelligent Robot

Arthur Gerstenfeld
Professor of Management
Worcester Polytechnic Institute
Worcester, MA 01609

Abstract

This paper describes an AI and robotics research project being conducted for NASA. The applications of our findings are for robots performing tasks in space.

The use of robots in the future must go beyond present applications and will depend on the ability of a robot to adapt to a changing environment and to deal with unexpected scenarios (i.e., picking up parts that are not exactly where they were expected to be). The objective of this research project was to demonstrate the feasibility of incorporating high level planning into a robot enabling it to deal with anomalous situations in order to minimize the need for constant human instruction.

Our heuristics can be used by a robot to apply information about previous actions towards accomplishing future objectives more efficiently. Our system uses a decision network that represents the plan for accomplishing a task. This enables the robot to modify its plan based on results of previous actions. Our system serves as a method for minimizing the need for constant human instruction in telerobotics.

This paper describes the integration of expert systems and simulation as a valuable tool that goes far beyond this project. Simulation can be expected to be used increasingly as both hardware and software improve. Similarly, the ability to merge an expert system with simulation means that we can add intelligence to the system.

This paper describes a satellite in space that has a malfunction. The expert system uses a series of heuristics in order to guide the robot to the proper location. This is part of task level planning.

The final part of the paper suggests directions for future research. Having shown the feasibility of an expert system embedded in a simulation, the paper then discusses how the system can be integrated with the MSFC graphics system.

Integration of Task Level Planning and Diagnosis for an Intelligent Robot

Arthur Gerstenfeld*

Introduction

It has been recognized for some time that intelligent telerobot architecture will become an increasingly important force in space activities [Reference 1]. There has been a great amount of previous research which has focussed on these issues [References 2-27]. At this point we are focussing on one part of the architecture, namely task level planning. The balance of this paper will describe our research in that area and some thoughts for future directions.

Project Objectives

1. The first objective was to show the feasibility of having a robot in space exhibit some autonomous behavior.

2. The second objective was to demonstrate the ability for the robot to use intelligent planning and replanning.

3. The third objective was to show how the system can be adaptable to different satellites or space station configurations.

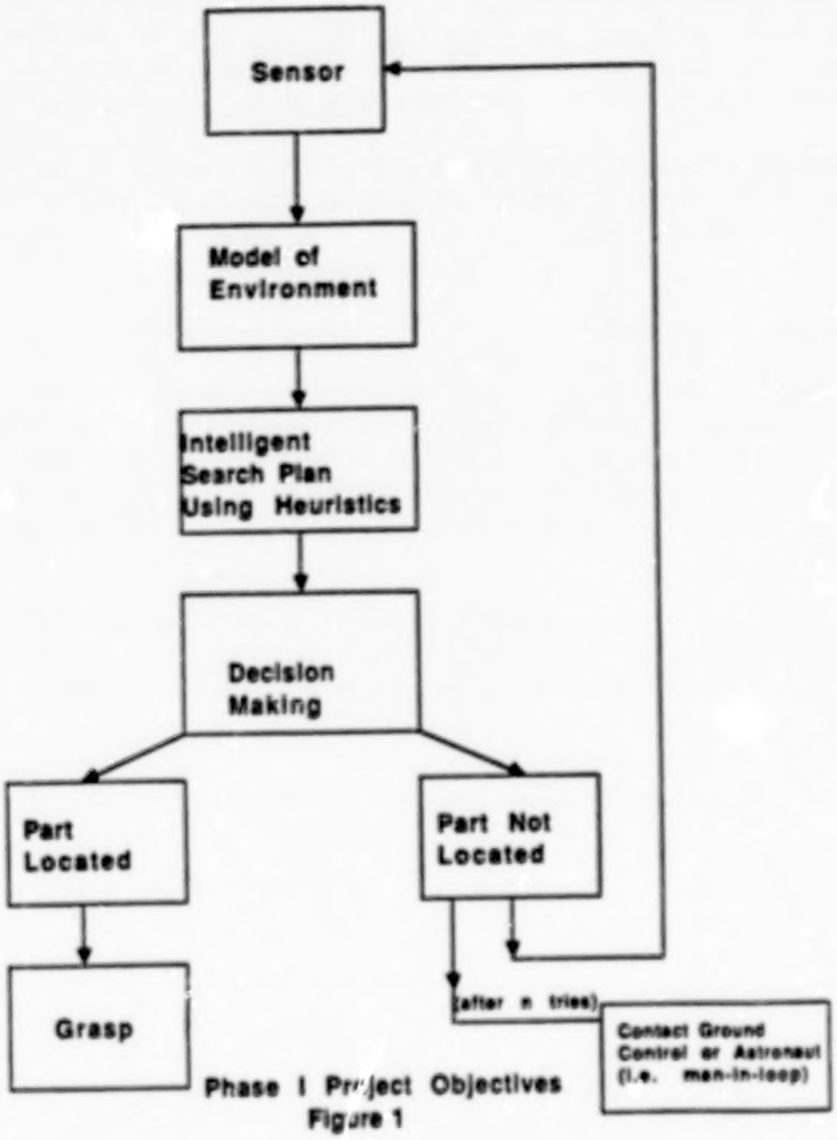
4. The fourth objective was keep the "man-in-the-loop" so that when the search did not yield the expected results, then ground control can redirect the search.

5. The fifth objective was to develop a feasibility model and demonstrate it at MSFC.

The objectives are shown graphically in Figure 1.

---

*Dr. Gerstenfeld received his Ph.D. from MIT and holds the position of Professor having an endowed chair at Worcester Polytechnic Institute (WPI). He is president of UFA, Inc. where the research described in this paper has (and is) taking place.


Sensor
Model of Environment
Intelligent Search Plan Using Heuristics
Decision Making
Part Located
Part Not Located
Grasp
(after n tries)
Contact Ground Control or Astronaut (i.e. man-in-loop)

Phase I Project Objectives
Figure 1

Work Carried Out

The work carried out can best be understood by referring to Figure 2. The robot is instructed to replace module A. The robot, however, is located at module H. Therefore, it must plan how to get from where it is located to the required location.

In order to do that planning we showed how it was necessary for the robot sensor to obtain two orientations:

1. Present location (i.e. "H")
2. Module directly above (i.e. "D")

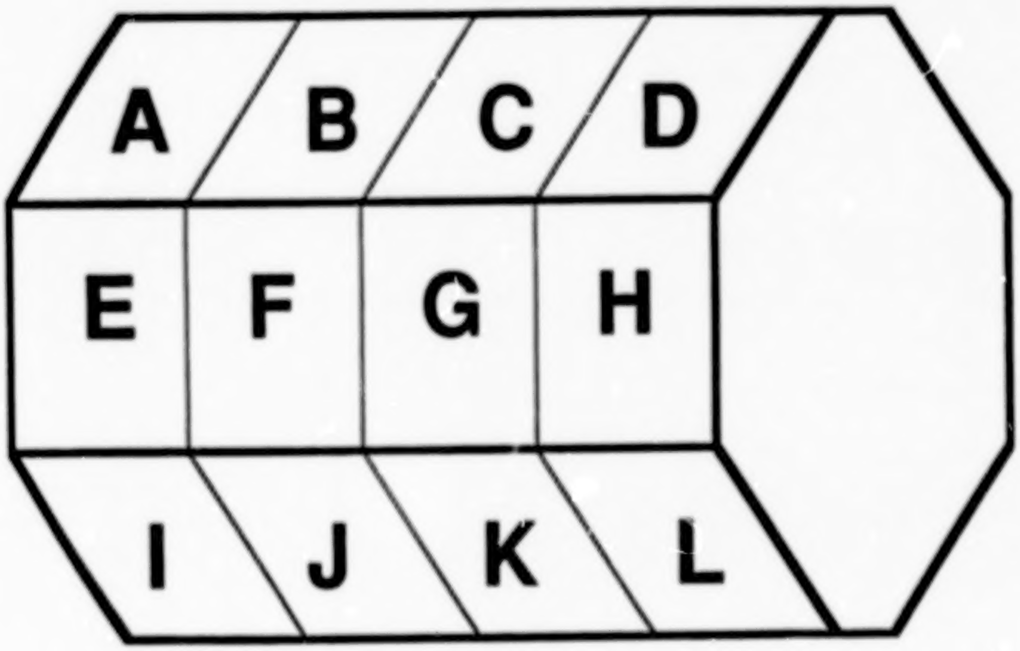
It is necessary for the robot to know the module above in order to gain orientation. This can be thought of as a person who is lost and he locates the street he is on but that is not enough. He must also know one other point (another street or landmark) in order to determine his orientation.

Having determined that the robot is at H and oriented so that D is directly above it - the decision can then be made that the robot must:

Move UP "1"
Move 4 to LEFT

After moving up 1 and 4 to left the robot may still not locate module A due to other discrepancies. In that case the robot will use further heuristic search using the same principle as described above. This can be done recursively until a solution is found or instructed otherwise by a human in the loop.

A second example is that the robot is at "G" and must move to "L". His orientation shows that "K" is above the robot and the robot then reasons that it must move 1 down and 1 to the right.



A Simulated Satellite with Different Modules

Figure 2

We, therefore, observe that each movement for the robot does not have to be given to it. Rather the robot can reason and perform intelligent search. During Phase I we designed the computer code necessary to accomplish independent search.

The work carried out included the building of a simulation on a COMPAQ 386. We designed the computer code to integrate the graphics and artificial intelligence.

Results Obtained

The results obtained can best be understood by referring to Figure 3 in terms of goals and subgoals. For example, in Figure 3 let us assume the goal is to locate a particular part of a satellite. This could also be a part on space station exterior or interior.

Figure 3 shows that:

- Subgoal 1 is "Recognize current location"

In order for the robot to regognize its present location it is necessary to have two further subgoals as follows:

- Subgoal 1.1 is "identify the place where the robot is currently located"

This is achieved by an action as follows:
"Use vision system to locate a point straight ahead".

This can best be thought of by thinking of a person lost in an area he does not know. The person must first look for a street sign to identify the street where he is standing. Having achieved the above it is necessary for the robot to find one other point. For the example we used, the other point was the module directly above the current module.

By using the analogy again of the lost person, once he found out what street he was located on he then had to locate one other marker. Let us assume that his eyes moved up and he located a cross street to the one on which he was standing. Referring to Figure 3 that is the following action:

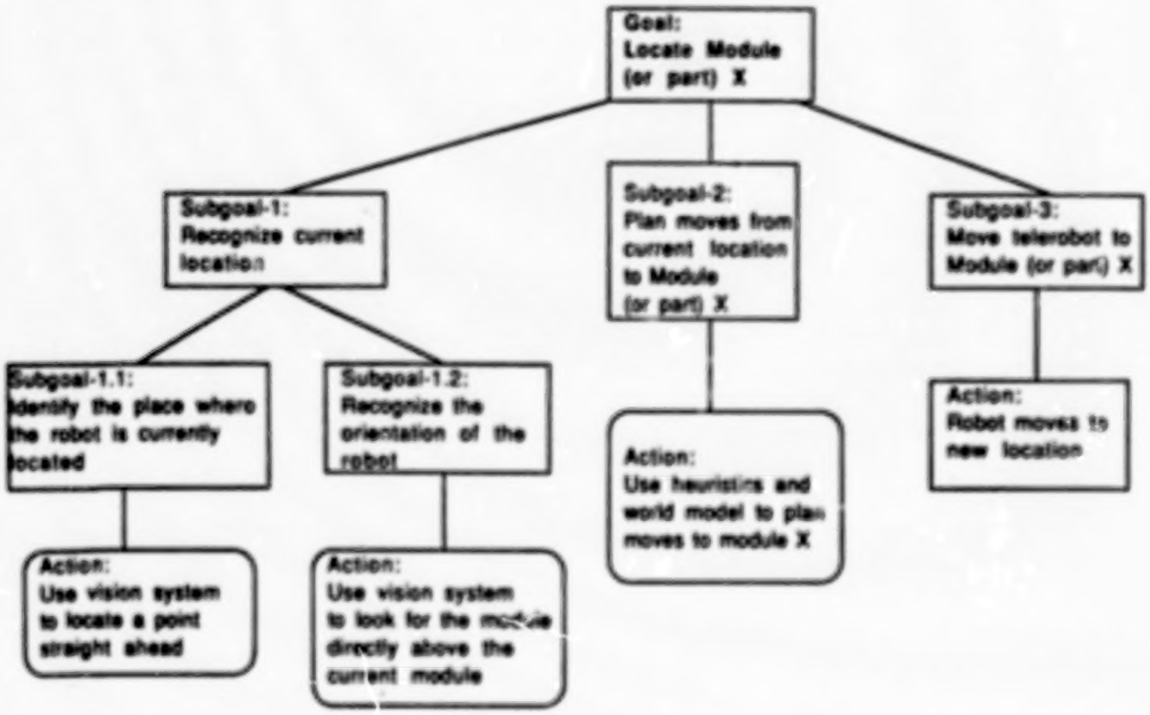
Use vision system to look for the module that is directly above the current module.

Having now established the locations of the robot (and its orientation) the subgoal 2 is as follows:

- Plan moves from current location to module (or part) X.

The action in that case is to:
Use heuristics and world model to plan moves to module X.

This can be thought of again as the lost person who has now sufficiently identified his location and orientation then planning his move to his destination.


Goal:
Locate Module (or part) X
Subgoal-1:
Recognize current location
Subgoal-2:
Plan moves from current location to Module (or part) X
Subgoal-3:
Move telerobot to Module (or part) X
Subgoal-1.1:
Identify the place where the robot is currently located
Subgoal-1.2:
Recognize the orientation of the robot
Action:
Use heuristics and world model to plan moves to module X
Action:
Robot moves to new location
Action:
Use vision system to locate a point straight ahead
Action:
Use vision system to look for the module directly above the current module

Different levels of decision-making in the system

Figure 3

The final subgoal is:

- Move telerobot to a particular location or part

The action associated with that goal is as follows:
Robot moves to new location

We showed this process of goals, subgoals, and actions using a model of a satellite. In this case we limited our investigation so that the input from the sensors were given by the user of the simulation. This can be visualized in Figure 4.

Future Directions

1. Development of an AI planner to generate task level path commands to a satellite servicer robot.

2. To integrate our system with the graphical simulation model at MSFC.

3. To include a model of a generic satellite with subsystems, e.g. power, altitude control, communications, etc.

4. To develop a diagnosis system that will be used to identify a list of possible failed subsystems.

5. To demonstrate higher level task planning by performing diagnosis and robot path planning in order to replace (or repair) a subsystem.

Our approach can best be understood by considering Figure 5. We shall be focussing on the task decomposition modules and show how they can perform real-time planning. The task decomposition modules plan and execute the decomposition of high level goals into low level actions.

Figure 4 shows on the right, the operator interface. On the left the global memory. The task (which might be "locate tool A" or "replace module B" is shown in H4 of Figure 1. Having received the task requirement the system would then check the world model by moving one square to the left in Figure 1 to M4. This is integrated with the sensory information shown in G4 for Figure 1.

The control system architecture is shown in Figure 5. The level we are focussing on is Level 4, which decomposes the object task commands specified in terms of actions performed on objects.

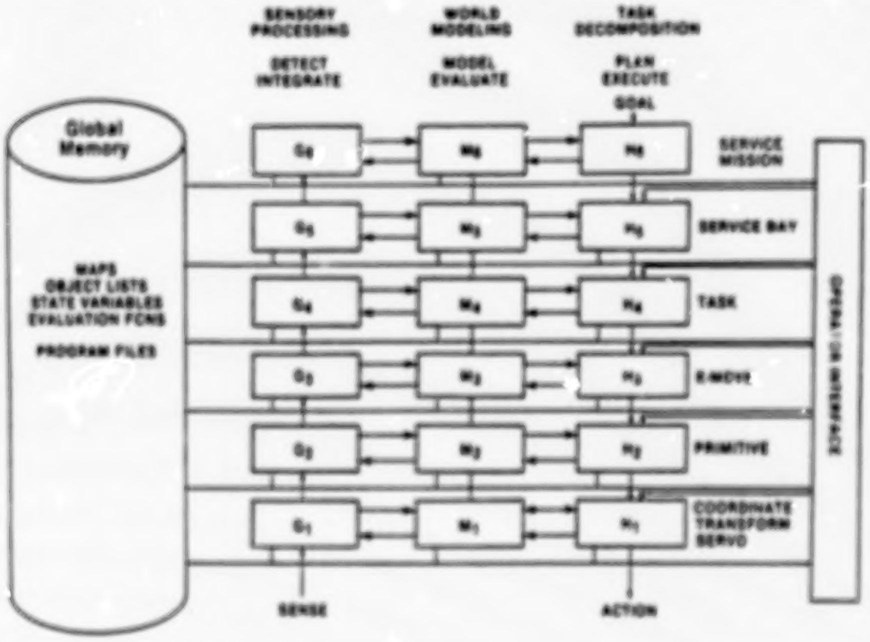


Fig.4 A hierarchical control system architecture for intelligent vehicles.

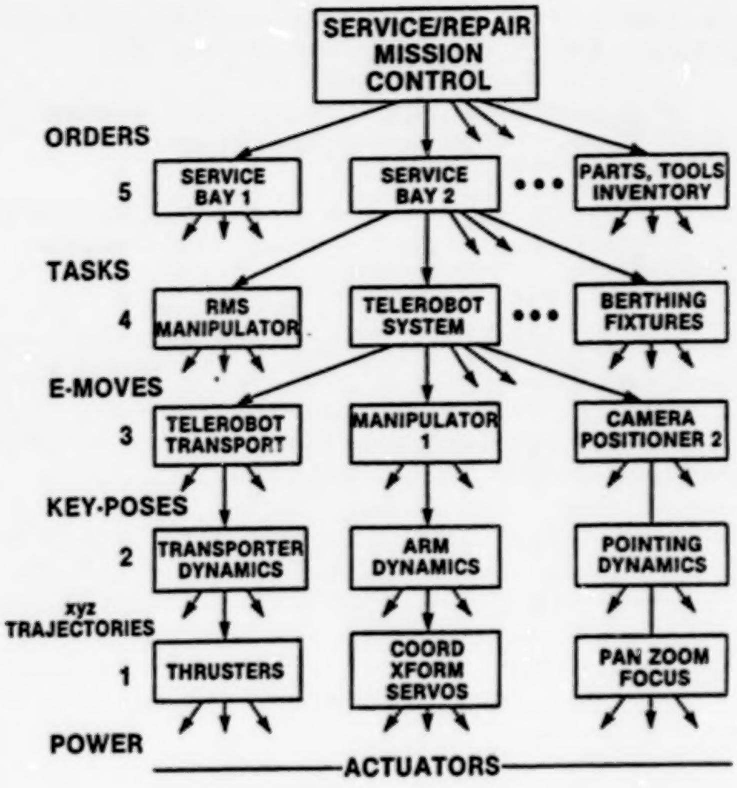


Figure 5

A six level hirearchical control system proposed for multiple autonomous vehicles.

References:

1. Albus, James S., Harry G. McCain, and Ronald Lumia, "NASA/NBS Std. Ref. Model for Telerobot Control System Architecture" (NASREM), U.S. Department of Commerce Technical Note 1235, 1987.

2. Berning, S., D.P. Glasson and G.A. Matchett, "Functionality and Architectures for an Adaptive Tactical Navigator System", Proceedings NAECON, Dayton, Ohio, May 1987.

3. Brooks, R.A., "Solving The Find-Path Problem by Good Represenation of Free Space," Proceedings of National Conference on A.I., pp. 381-386, 1982.

4. Cohen, P.R., Heuristic Reasoning About Uncertainty: An Artificial Intelligence Approach, Potman, London, 1985.

5. Cromarty, A.S., D.G. Shapirot, and M.R. Fehling, "Still planners run deep: Shallow reasoning for fast replanning," Proceedings SPIE Conference on Applications of Artificial Intelligence, No. 485, pp. 138-145, 1984.

6. Crowley, J.L., "Dynamic World Modeling for an Intelligent Mobile Robot," Proceedings of 7th International Conference on Pattern Recognition, pp. 207-210.

7. Delaney, J.R., R.T. Lacoss, and P.E. Green, "Distributed Estimation in the MIT/LL DSN Testbed," Proceedings American Control Conference, pp. 305-311, San Francisco, CA, June 22, 1983.

8. Duda, R.O., P.E. Horb, and N.J. Nasson, "Subjective Bayesian Methods for Rule-Based Inference Systems," Technical Note 124, AI Center, SRI International, Menlo Park, CA, 1976.

9. Erman, F., Hayes-Roth, V.R. Lesser, and R.D. Reddy, "The HEARSAY-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty," ACM Computing Surveys, Vol. 12, pp. 213-253, 1980.

10. Evers, D.C., D.M. Smith, and C.J. Staros, "Interfacing an Intelligent Decision Maker to a Real-Time Control System", Proceedings SPIE on Applications of Artificial Intelligence, Vol. 485, pp. 60-64, 1984.

11. Gerstenfeld, A., "Social and Economic Effects of Industrial Robots", Proceedings of International Conference on Industrial Robots, Paris, France, 1982.

12. Girealt, G., R. Sobek, and R. Chatila, "A Multi-Level Planning and Navigation System for a Mobile Robot; a First Approach to HILARE," Proceedings of 6th International Joint Conference on A.I., pp. 335-337, 1979.

13. Glasson, D.P., and J.L. Pomarede, "Adaptive Tactical Navigation-Phase II Concept Development," Report No. AFWAL-TR-86-1066, The Analytic Sciences Corporation, Reading, MA, September 1986.

14. Green, P.E., "Resource Control in a Real Time Target Tracking Process," Proceedings Fifteenth Asilomar Conference on Circuits, Systems, and Computers, pp. 424-428, Pacific Grove, CA, November 9, 1981.

15. Green, P.E., "Distributed Acoustic Surveillance and Tracking," Proceedings Distributed Sensor Networks Workshop, pp. 117-141, M.I.T. Lincoln Laboratory, Lexington, MA, January 6, 1982.

16. Green, P.E., "AF: A Framework for Real-Time Distributed Cooperative Problem Solving," Presented at the 1985 Distributed AI Workshop, Sea Ranch, CA, Artificial Intelligence Research Group, Worcester Polytechnic Institute, Worcester, MA, November 1985.

17. Green, P.E., "The Activation Frame Method for Real-Time Expert Systems," Technical Report EE85PG04, Department of Electrical Engineering, Worcester Polytechnic Institute, Worcester, MA, October 12, 1985.

18. Green, P.E., "Issues in the Application of Artificial Intelligence Techniques to Real-Time Robotic Systems," Proceedings 1986 ASME Computers in Engineering Conference, Chicago, IL, July, 1986.

19. Green, P.E., "Resource Limitation Issues in Real-Time Intelligent Systems," Proceedings SPIE Conference on Applications of Artificial Intelligence III, Vol. 635, Orlando, FL, April 1986.

20. Green, P.E., "Working Paper on the Incremental Evidence Technique", Technical Report EE86TAIRGO3, Worcester Polytechnic Institute, Worcester, MA, September 1986.

21. Green, P.E., "Real Time Intelligent Issues in the Development of the Adaptive Tactical Navigator," Proceedings of SOAR Conference, Johnson Space Center, 1987.

22. Green, P.E., "AF: A Framework for Real-Time Distributed Cooperative Problem Solving," Collected Paper of the 1985 Distributed AI Workshop, Sea Ranch, CA, pp. 337-356, November 1985.

23. Green, P.E. "Resource Control in a Real-Time Target Tracking Process," Proceedings Fifteenth Asilomar Conference on Circuits, Systems and Computers, pp. 424-428, Pacific Grove, CA, November 9, 1981.

24. Greer, T.H., "Artificial Intelligence: A New Dimension in EW," Defense Electronics, pp. 108-128, October 1985.

25. Ichikawa, Y., and N. Ozaki, "Autonomous Mobile Robot," Journal of Robotic Systems, Vol. 2 (1), pp. 135-144, 1985.

26. Jones, H.L. and A.L. Pisano, "Adaptive Tactical Navigation: Report No. AFWAL-TR-85-1015, the Analytic Sciences Corporation, Reading, MA, April 1985.

27. Kiersey, D.M., J.S. Mitchel, D.W. Payton, and E.P. Preyss, "Multilevel path planning for autonomous vehicles," Proceedings SPIE Conference on Applications of Artificial Intelligence, No. 485, pp. 133-137, 1984.

N89-15561

A Graphical, Rule Based Robotic Interface System

James W. McKee
University of Alabama, Huntsville
Box 212, RI-A4
Huntsville, Alabama 35899

John Wolfsberger
NASA/MSFC
EB 42
Huntsville, Alabama 35812

ABSTRACT

The ability of a human to take control of a robotic system is essential in any use of robots in space in order to handle unforeseen changes in the robot's work environment or scheduled tasks. But in cases in which the work environment is known, a human controlling a robot's every move by remote control is both time consuming and frustrating to the human.

A system is needed in which the user can give the robotic system commands to perform tasks but need not tell the system how to perform the tasks. To be useful, this system should to be able to plan and perform the tasks faster than a telerobotic system. The interface between the user and the robot system must be natural and meaningful to the user.

This paper describes a high level user interface program under development at the University of Alabama, Huntsville. The authors propose in this paper a graphical interface in which the user selects objects to be manipulated by selecting representations of the objects on projections of a 3-D model of the work environment. The user may move in the work environment by changing the viewpoint of the projections.

The interface uses a rule based program to transform user selection of items on a graphics display of the robot's work environment into commands for the robot. The program first determines if the desired task is possible given the abilities of the robot and any constraints on the object. If the task is possible, the program determines what movements the robot needs to make to perform the task. The movements are transformed into commands for the robot. The information defining the robot, the work environment, and how objects may be moved is stored in a set of data bases accessible to the program and displayable to the user.

Introduction

The graphical user interface, to be described in this paper, is part of a project to develop a software system that will enable users to control robots from a task level instead of having to either teach the robot the path or write programs in the robot's language or when using simulation programs specify points in the works space and actions to be performed [3].

There are two objectives of this project which have a strong influence on the requirements of the graphical user interface. The first objective is to divide the software into functional modules such that new versions of any module may be "plugged in" and the system tested. The second objective is to be able to incorporate into the system the knowledge and expertise that a person usually needs to have to create programs for the robot.

We have divided this project into four modules: user interface, path planning, environment calibration, and robot code generation.

The user interface module will contain the graphical descriptions and all the knowledge, rules, and constraints about the robot and the work space. The function of a robot is to move objects. The user interface is the means by which the user tells the system which objects are to be moved and where. The graphical user interface being presented in this paper is only one of many possible user interface modules.

From the robot task requirements and the given geometric and dynamical constraints on the robot motion, the path planning module will define a path in the robot work space that will avoid collisions and satisfy the constraints on the joint dynamics.

The environment calibration module will allow the robot to calibrate itself to a task board or other objects in the work space. The module will also allow the system to verify that what the robot "sees" in the work space is what it should see.

The robot code generation module transforms the internal motion representation data into movement commands for a particular robot.

Graphical User Interface

The user interface module has been divided into three projects: object definition user interface (ODUI), object movement user interface (OMUI), and rule based task planner (RBTP). The ODUI program allows the user to create new objects and enter the objects into the system data base. The

OMUI allows the user to move around in the work space and select objects to be moved. Once the user selects an object and its destination, the RBTP determines if the object can be moved and if so makes a list of fixed and flexible paths.

To support the graphics requirements on this project, the user interface software is being developed on a Silicon Graphics 3020 graphics station. The Silicon Graphics computer is connected to a PUMA 562 robot by a RS232 line and running the DDCMP protocol. This will be used as the hardware configuration for system tests.

Object definition user interface

The purpose of the ODUI software is to allows a user to create objects in the system. An object is a set of data that contains the following types of information: name, geometric description, physical attributes, movement and positional constraints, and construction list.

The objective of this module is to make it easy for a user to create the graphical description of objects that will form the environment or are to be moved. The graphical information will be used by the OMUI, the path planner module and the environment calibration module. This is also how the user creates the knowledge base of the physical attributes and movement constraints of the object that the expert system will need to determine if and how objects are to be moved. The software being developed is intended to be a flexible framework by which to store the knowledge data. This software itself does not interpret or process any of the knowledge data.

The user interface consists of levels of mouse selectable pop-out menus buttons. The top level menu allows the user to examine objects, edit objects, create objects, or delete objects.

Objects are essentially data bases. To examine an object, the user views the various lists of data in the objects data base. This could be by viewing projections of the graphical representation of the object or by viewing, in text, format the contents of the other list of the object.

Each object is created with its own local coordinate system. All references to the position of the object are with respect to the origin of the object's coordinate system. The object's geometric description and positional constraints are defined in this coordinate system.

Each object has a unique name. This is the key by which the object is referenced when used to form another object or when moved. Any object can be used as a template to create copies of itself. The user must supply a unique name for the new object when creating a new object from a template.

One or more objects may be combined with or without added graphics to create a new object. When an object is incorporated into a new object, the name of the object being incorporated and its rotation and translation are placed in the new object's construction list.

The user through a process of creating objects and combining objects builds up the robot, the environment and the robot work space. Starting from the top and working down, the robot work space contains everything and is an object which is composed of two objects: the environment and the robot. The robot is an object composed of objects that are the links of the robot and a data base of the kinematic equations for the links.

The environment is composed three types of objects: movable objects, receptacle objects, and the task board object. Movable objects are objects that have been selected by the user to be movable by the robot. Receptacle objects are objects in which or on which movable objects may be placed. Movable objects may only be moved from one receptacle object to another receptacle object.

The task board is everything else in the environment. The task board object is needed for the geometric description it generates in the path planner module and the environment calibration module and to give the user a geometric feel for where the other objects are located.

Receptacle objects contain the information about how and where movable objects may be placed. For example, a receptacle object could be a hole. In this case the hole would carry the information about the size and shape of a movable object that could be placed in the hole and the fact that the object must be inserted. Another type of receptacle object would be a pad. A pad would contain the information that a movable object could be placed on it. The pad could also contain the information about the orientation of the movable object when it placed on the pad. More than one receptacle object may be placed at the same geometric location, each designed for a particular class of movable objects.

Graphical descriptions are created by the user "drawing" simultaneously in three orthogonal projection windows. The size and position of the windows on the monitor is user controllable. Descriptions are created by combining volumes and surfaces. There is a set of primitive volumes that include cones, cylinders, and rectangles. These primitives may be stretched to whatever size is needed. Surfaces are planar polygons. The user combines these volumes and surfaces to create graphic descriptions of the objects.

Object movement user interface

The objective of the OMUI software is to allow the user to select objects and indicate where they are to be moved in the robot's work space. The user sees on the graphics monitor a two-dimensional projection of the three dimensional work space of the robot.

The user may translate, rotate, and zoom the work space. The user may select one of these three modes of moving the work space by mouse selectable menu buttons on the side of the screen. Once an option has been selected, the user controls the direction of motion of the work space by pressing one or more of the buttons on the mouse and the rate of motion by the motion of the mouse.

The cursor mode is another mouse selectable menu button. Once the cursor mode is selected, the user may select what information about objects will be displayed as the curser moves over their projection. The user may turn on object highlighting, object name display, and/or object data display. The highlighting switch causes the movable objects and receptacle objects to be highlighted when the cursor moves onto them. The object name switch causes the name(s) of the object(s) under the cursor to be displayed. The object data display switch allows the user to examine any of the knowledge data for selected objects.

Once the work space is in the desired orientation and magnification, the user may move a curser around on the surface of the projection. After the cursor mode is selected, the cursor is moved by pressing the right button on the mouse as the mouse is moved. A movable object is grabbed by clicking the middle button of the mouse when the cursor is on the object. Only movable objects may be grabbed. The destination is selected by moving the cursor onto a receptacle object and clicking the left mouse button. The object is dropped if the left button is clicked anywhere else, in which case there is no change in the geometric configuration.

At the present time only one object movement can be selected in a session. A future enhancement will be to be able to create lists of object movements that are to be carried out in succession with the environment being updated after each object movement.

Rule based task planner

Once a movable object has been grabbed and a destination receptacle object selected, the RBTP must determine if it is possible to move the object. The RBTP generates a set of knot points for the geometric path of the movable object. A knot is a pose (x, y, z, roll, pitch, yaw) through which the movable object is to pass. Connecting the knot points are fixed and flexible paths.

A fixed path is a path on which the pose of the movable object is defined along the whole path. On a flexible path the pose of the object is defined only at the end points. The RBTP will also collect from the knowledge base of the object a set of any applicable constraints on the movement of the object. Constraints could be items such as a maximum acceleration, allowable tilt angles on the object, maximum gripping pressure, etc.

From the information in the data bases of the source and destination receptacle objects, the RBTP creates the set of knots and the fixed paths. For example, assume the source receptacle object was a pad, the movable object was a peg, and the destination receptacle object was a hole. Then the path of the object could consist of a fixed path, a flexible path and a fixed path. The first fixed path would be the path to pick the peg up off the pad. The second fixed path would be the path to insert the peg into the hole to the desired depth. And the flexible path would be between the end of the first fixed path and the start of the second fixed path.

For each flexible path, the path planning module will create a near optimum, collision-free path that does not violate any of the dynamic constraints on the joints of the robot or any of the constraints on the movement of the object.

Since we are building a software system, the expert system software must be capable of being incorporated into the overall software. Therefore, it was decided to have the expert system run on the Silicon Graphics computer. Although LISP is available on the Silicon Graphics, it was decided to use CLIPS. CLIPS is a forward chaining expert system shell written in C. The source code for CLIPS is commercially available [1]. Harrington [2] has compared CLIPS, LISP, Prolog, and OPS5 and has concluded "Because of its embedability, its expandability, and its smaller size, CLIPS would be the better selection for embedding low-level ES capability within a control system".

Conclusions

This paper has presented an overview of the software being developed at the University of Alabama, Huntsville to enable a user to control a robot from a task level. The main emphasis of the project is to develop a set of software modules that work together as a system. This presentation has concentrated on the user interface portion of the project and how the requirements of the overall system have affected the design of the user interface portion.

Acknowledgements

Research for this paper has been supported in part by a grant form the Science, Technology and Energy Division of the Alabama Department of Economic and Community Affairs. However, any opinions, findings, conclusions or recommendations expressed herein are those of the authors and do not necessarily reflect the views of ADECA and the State of Alabama.

References

[1] Giarratano, J. C., CLIPS User's Guide, CLIPS Reference Manual, COSMIC Program # MSC-21208, 382 E. Broad St. Athens, GA, 30602.

[2] Harrington, J. B., "CLIPS as a Knowledge Based Language," Third Conference on Artificial Intelligence for Space Applications, Huntsville, November 2-3, 1987, pp.33-40.

[3] Mckee, J. W. and Wolfsberger, J., "High Level Intelligent Control of Telerobotic Systems," Conference on Space and Military Applications of Automation and Robotics, June, 1988, Huntsville.

N89-15562

PDA: A Coupling of Knowledge and Memory for Case-based Reasoning

S/ Bharwani*, J. Walls and E. Blevins

Martin marietta Manned Space Systems
MSFC, Huntsville, AL

Topic: Computer Advisor

Keywords: Conceptual Memory, Casual Dependency, Similarity

Abstract

There is little doubt about the role of knowledge in autonomous intelligent reasoning. Problem solving in most domains requires reference to past knowledge and experience whether such knowledge is represented as rules, decision trees, networks or any variant of attributed graphs. Regardless of the representational form employed, designers of expert systems rarely make a distinction between the static and dynamic aspects of the system's knowledge base.

The current paper clearly distinguishes between knowledge-based and memory-based reasoning where the former in its most pure sense is characterized by a static knowledge base resulting in a relatively brittle expert system while the latter is dynamic and analogous to the functions of human memory which learns from experience.

The paper discusses the design of an advisory system which combines a knowledge base consisting of domain vocabulary and default dependencies between concepts with a dynamic conceptual memory which stores experiential knowledge in the form of cases. The case memory organizes past experience in the form of MOPs (memory organization packets) and sub-MOPs. Each MOP consists of a context frame and a set of indices. The context frame contains information about the features (norms) that are common to all the events and sub-MOPs that are indexed under it.

Case memory is designed to achieve maximal clustering of cases by similarity of important features. The memory is also self-organizing in the sense that it strives to minimize the search effort for the retrieval of relevant cases. It accomplishes this by identifying similarities between indices in terms of the order (first order, second order, logarithmic,

PRECEDING PAGE BLANK NOT FILMED

etc.) of the response models and merging them into generalized sub-MOPs when possible.

Problem solving is accomplished by either referring to a past similar case if one is found in memory, or by triggering the acquisition missing knowledge. New knowledge is acquired by a carefully controlled design of experiments followed by analysis and statistical modeling of results to derive casual relationships between concepts. The models are represented within the cases that provide the context in which such models are relevant and hence applicable.

N89-15563

Approximate Spatial Reasoning*

Soumitra Dutta

Computer Science Division
University of California
Berkeley, CA 94720

Abstract

It is a truism that much of human reasoning is approximate in nature. Formal models of reasoning traditionally try to be precise and reject the fuzziness of concepts in natural use and replace them with non-fuzzy scientific explicata by a process of precisiation. As an alternate to this approach, it has been suggested that rather than regard human reasoning processes as themselves "approximating" to some more refined and exact logical process that can be carried out with mathematical precision, the essence and power of human reasoning is in its capability to grasp and use inexact concepts directly. This view is supported by the widespread fuzziness of simple everyday terms (e.g., near, tall) and the complexity of ordinary tasks (e.g., cleaning a room). Spatial reasoning is an area where humans consistently reason approximately with demonstrably good results. Consider the case of crossing a traffic intersection. We only have an approximate idea of the locations and speeds of various obstacles (e.g., persons and vehicles crossing the intersection), but we nevertheless manage to cross such traffic intersections without any harm. The details of our mental processes which enable us to carry out such intricate tasks in such an apparently simple manner are not well understood. However, it is important that we try to incorporate such approximate reasoning techniques in our computer systems. Approximate spatial reasoning is very important for intelligent mobile agents (e.g., robots), specially for those operating in uncertain or unknown or dynamic domains. In such situations, several factors make the use of approximate reasoning techniques imperative:

[1] It may be difficult or sometimes even impossible to collect precise information about the environment.

---

* This work has been partially supported by NASA Grant NCC-2-275

[2] Most real world robots have to operate in the face of real constraints such as limited memory and time for collecting observations/making inferences.

[3] Many real world environments are hostile in the sense that they are dynamic, uncertain and often hazardous. In such situations, the agent should be able to deal effectively with sudden stimuli presented by the environment.

In this paper we present a model for approximate spatial reasoning using fuzzy logic to represent the imprecision in the environment. We develop algorithms to reason from approximate spatial information such as:

A is about 5 miles away and is coming towards me quite fast.

B is quite near C, but is far north of D.

B is moving much faster than C.

The kind of spatial information dealt with has two notable characteristics: it is approximate and may be incomplete. We report on algorithms that can be used to reason spatially from such approximate information. This is our ongoing research and thus we report initial results. In particular we present algorithms for determining approximate relative positions of objects, in both static and dynamic domains. These algorithms have the attractive features of being both formally adequate (i.e., complete and consistent) and computationally tractable (polynomial time). This is a first step towards the formulation of approximate path planning algorithms. We foresee tremendous applications of such approximate reasoning methods in robots operating in both space and on earth. It will may be the case that these approximate reasoning methods hold the ultimate solution to tackling the mind-boggling complexity of the real world which now limit the performance of these robots.

N89-15564

REPRESENTATION AND MATCHING OF KNOWLEDGE TO DESIGN DIGITAL SYSTEMS

J. U. Jones
Rockwell International
555 Discovery Drive
Huntsville, Alabama 35805

S. G. Shiva
Computer Science Department
University of Alabama in Huntsville
Huntsville, Alabama 35899

ABSTRACT

In this paper, we describe a knowledge-based expert system that provides an approach to solve a problem requiring an expert with considerable domain expertise and facts about available digital hardware building blocks. To design digital hardware systems from their high level VHDL (Very High Speed Integrated Circuit Hardware Description Language) representation to their finished form, a special data representation is required. This data representation as well as the functioning of the overall system is described.

1. INTRODUCTION

Automatic hardware synthesis is important, because the complexity of integrated circuits has became as high as millions of transistors per device. Yet, the turn around time available for each design has become shorter and shorter due to competition. Design aids such as logic minimization tools, wire routers, simulation tools, and hardware synthesis systems have been developed; but they do not approach the efficiency of manual hardware design. Overviews of hardware synthesis systems can be found in [2,6].

The Department of Defense initiated the Very High Speed Integrated Circuit (VHSIC) Program to aid in the production of military integrated circuits [7]. To create an integrated design tool taking the designer through all phases of development, testing and evaluation, the VHSIC Hardware Description Language (VHDL) was developed [8,9]. Research in the area of developing an expert system which would function as an integrated design tool has been done at the University of Alabama in Huntsville (UAH) by Green [3] and Klon [4]. Green designed the University of Alabama Hardware Expert Synthesis System (UHESS) which serves as a prototype design consultant in the selection of VHSIC chips. Klon investigated the issues concerning interfacing UHESS to VHDL. He extended Green's knowledge base representation data structure and named it a hologram, which was used to synthesize sample hardware designs. This paper extends Klon's hologram representation.

2. THE EXPERT SYSTEM

To design digital hardware, first a VHDL source code is developed and input into the expert system. The expert system translates this source code into a hologram representation. The design library contains holograms of modules that have been designed earlier. These holograms can be at different levels of sophistication (detail). They may contain only one gate, a combi-

N89-15564

national circuit, a complex circuit like a printer interface board, or a whole microcomputer.

Once a hologram representation is given, the inference engine can begin to search the library for similar ones. The outcome of this search could be: a matching hologram found, a similar hologram or holograms are found, or a similar hologram does not exist. In each case the actions to be taken to define subholograms are described later in this paper.

When hologram representations are defined for the subholograms, the procedure described above can be used recursively until, the top hologram description is decomposed into a tree with nodes of subholograms where the leaf node holograms can be found in the library. The phase just described is the requirement decomposition phase. The next phase is the synthesis. Here children nodes are synthesized to become the parent hologram. This process begins at the leaf nodes and propagates actual design constraints upward on the tree until the root node is reached.

The difficulty lies in the fact that design uses nonmonotonic reasoning. The correctness of design assumptions can be verified only by exploring all consequences of the assumptions. Wrong assumptions can be taken during the analysis as well as the synthesis phase going from lower level nodes toward higher level subholograms. The source of incorrect decisions can be pinpointed and another design assumption has to be taken. All consequences of the previous incorrect decisions have to be traced and replaced with the consequences of the new decision. Therefore, the synthesis process can be described as a constant back and forth motion between analysis and synthesis, but eventually will end up at the root node. At that point the finished circuit design is output and incorporated into the library. Figure 1 shows the digital circuit design process.

Another difficulty is that in digital circuit design there is no one best design; there may be several equivalent good designs. There is a question wheather alternative designs are better. Or if they are better, what is it that makes them better? There are several options a design could be optimized for. Number of components, cost, speed, size, heat generated, and long life are a few examples. To optimize a design to any of these requirements or any combination of them requires a different approach. This problem could be solved two ways: one is that the hardware synthesizer develops all alternatives with a nondeterministic design approach; the other is that the physical device constraints are taken into account early on in the design. It is easily seen that the nondeterministic design approach would take many iterations, and in most cases would be inefficient. Therefore, the purely top down design methodology can not be used, and a combination of top down and bottom up design methodology is needed. Some actual device constraints have to be taken into consideration at the design specification (VHDL code generation) phase. This implies that the way the specification is written in VHDL has a very large effect on the implementation.

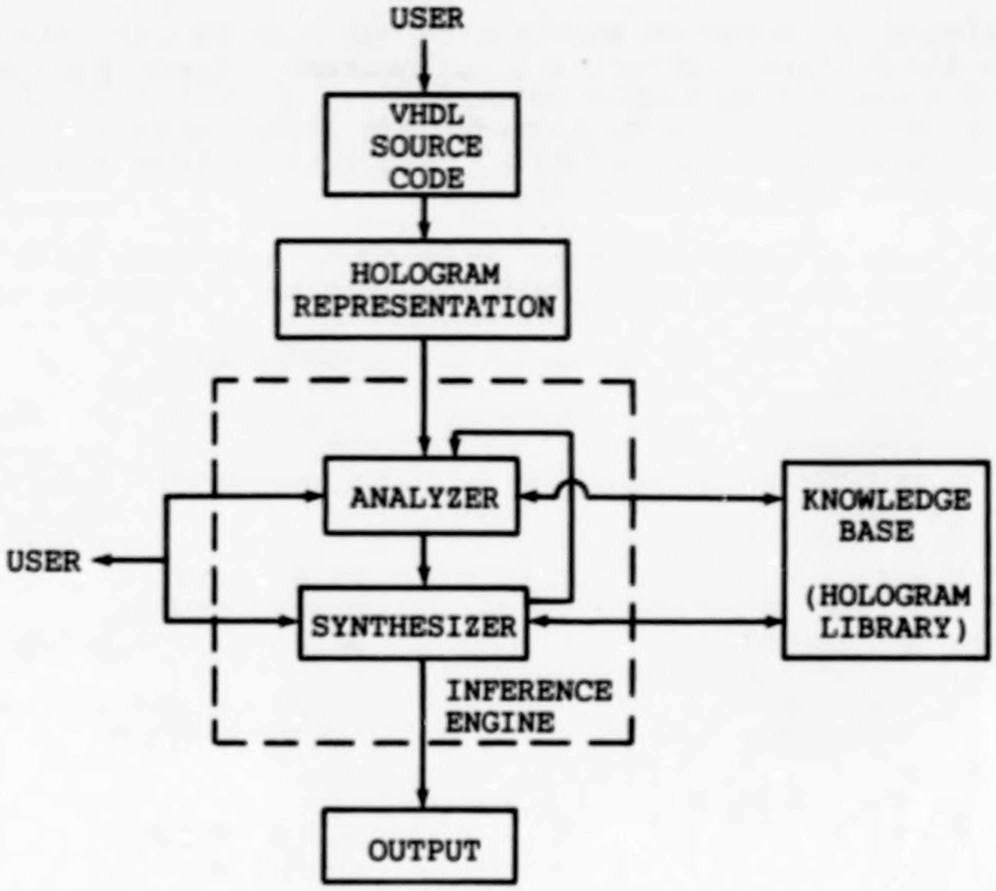


Figure 1. The digital circuit design process.

3. HOLOGRAM DATA STRUCTURE AS A DESIGN REPRESENTATION

Green's dissertation focused on the Knowledge Base (KB) representation for an Expert System which would design digital circuitry. It investigated issues such as knowledge acquisition tasks, the knowledge being represented, and proposed a knowledge representation scheme. This knowledge representation scheme combined frames and production rules to generate prototypes. This representation scheme was selected because it integrated the strengths of both representations. The prototype representation contains slots for knowledge storage. This knowledge consists of data pertaining to VHSIC chips and rules establishing default values and knowledge heuristics.

Klon investigated the issues of interfacing UHESS to VHDL and developed a scheme to represent dynamic semantics. The hologram which is a data structure to represent one or a collection of devices combines the features of prototypes and semantic networks. Modules (a collection of chips with their interfaces and functions) represent objects, functions, and activities, while signals are carriers of information between modules. Hierarchical semantic structures are propagated through modules; and semantics of the connectivity between modules are propagated through signals. Therefore, there is a need for two types of prototypes: module and signal. Holograms are a network of modules and signals. Figure 2 shows an abbreviated hologram for an ALU. The hologram description contains a hologram name, type, port assignments, element (or submodule description), a netlist which shows how the elements are interconnected, and rules containing information on the use of the hologram. Figure 3 depicts the abbreviated hologram structure for the ALU.

The hologram representation holds advantages over pure prototype or pure semantic network representation. Comparison of two

```
NAME:  ALU                              NAME:  BUS
TYPE:  MODULE                           TYPE:  SIGNAL
IN:  A, BUS                             DESCRIPTOR:  8, BIT
     B, BUS                             CONDITIONS:  5, MA., MAX
     S, BIT                             HEURISTICS:  RULE062
     CLK, BIT                                        RULE065
     COMP, BIT
OUT:  C, BUS                            NAME:  BIT
LOCAL:  Y, BUS                          TYPE:  SIGNAL
ELEMENT ASSIGNMENTS:                    MODIFIER:  TERMINAL
      1, EIGHT_BIT_ADDER
      2, SHIFTER                        NAME:  EIGHT_BIT_ADDER
      3, TWOS_COMP                      TYPE:  MODULE
NETLISTS:                               ...
      A; (1,1), (2,2)
      B; (3,2)                          NAME:  SHIFTER
      S; (2,3)                          TYPE:  MODULE
      CLK; (2,1), (3,1)                 ...
      COMP;  (3,3)
      C;  (1,3), (2,4)                  NAME:  TWOS_COMP
      Y;  (1,2), (3,4)                  TYPE:  MODULE
HEURISTICS:  RULE071                    ...
             RULE076
```

Figure 2. Abbreviated hologram for an ALU. [4]

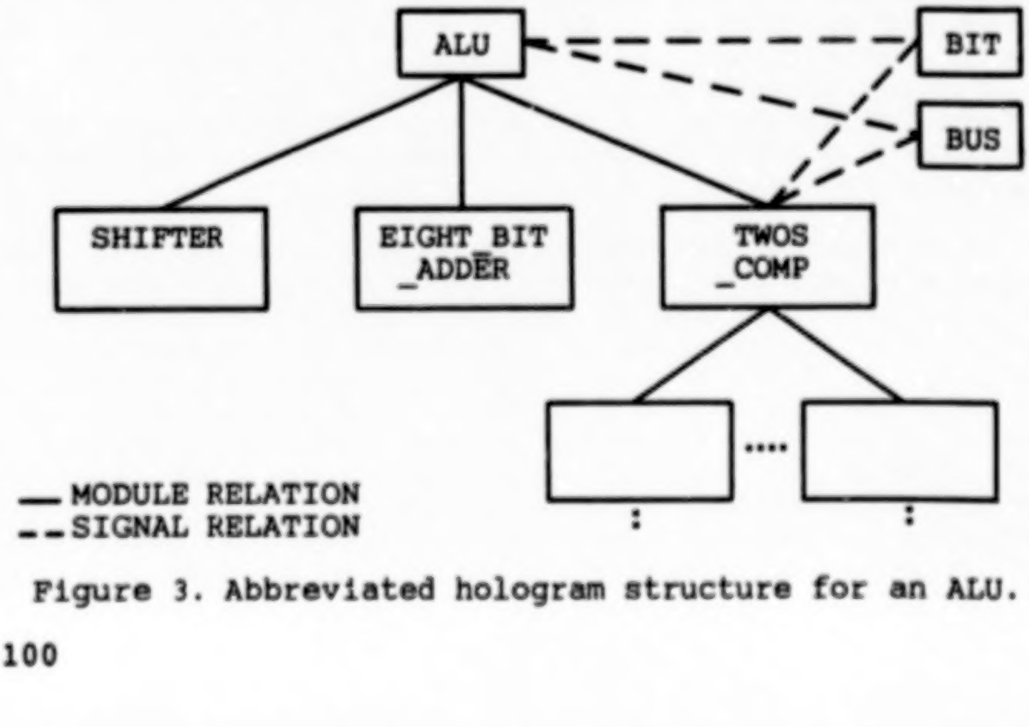


Figure 3. Abbreviated hologram structure for an ALU. [4]

semantic nets is slow since the whole network has to be traversed. But new meanings can be easily created. Pure prototype representation does not facilitate easy creation of new meanings; but comparison is easy since only the root modules and signals need to be compared. The hologram provides mechanisms for both the dynamic creation of new meanings and easy comparison.

To extend this work several designs were implemented from a hologram representation to a finished design. During these investigations, properties of the hologram required for easy automatic synthesis were developed. For automatic hardware synthesis the hologram data structure has to facilitate easy decomposition, easy similarity assessment, and should be a vehicle for name unification. (Similar modules may have different port names, and the number of ports do not have to match.) To facilitate these requirements, the hologram should contain slots of information pertaining to the detailed functioning of the hologram and attribute slots for easy determination of similarity.

3.1. Decomposition

For easy automatic decomposition and name unification the function of the hologram should be in a special form. Consider the functioning of a digital circuit. Here, the inputs are transformed to intermediate signals by a function, and these intermediate signals are transformed to the next level of intermediate signals by another function, and so on until the output signals are generated. Consider the analogy between these functions and primitives of a language. Then a language comes to mind whose primitives are the functions of actual hardware devices. Let us call this language the Actual Device Language (ADL). If the function of a hologram can be expressed in this language, the hardware design is done, in theory (on a high level), because it does not take into consideration lower level design constraints (such as propagation delay, power supply requirement etc...). Each leaf node in the design tree is an Actual Design Primitive, each intermediate node is an aggregate of Actual Design Primitives having as many subfunctions as there are submodules. Therefore, each function definition at a higher level is expressed as a number of subfunctions, where the subfunctions have subfunctions and so on.

In the decomposition or analysis phase, holograms are compared to library holograms, and if no matching (or similar) hologram is found, the hologram is simply decomposed to its subholograms. Figure 4 shows selected Actual Device Primitives, and Figure 5 depicts a hologram function which can be decomposed into subfunctions. As can be seen from the examples, the Actual Device Primitives are procedure-like and readable. Actual signal names take the place of ENABLE or INPUT1.

3.2. Similarity Assesment

The data model for the knowledge base is a hyper-semantic data model. An overview of traditional and semantic data models can be found in [5]. To compare holograms to library holograms each hologram contains slots for attributes. The collection of a., buffer

```
    INPUT --→ OUTPUT
```

b., two-to-one multiplexer

```
    IF      (ENABLE AND CLOCK)     THEN INPUT1 --→ STORE --→ OUTPUT
    ELSE IF (NOT ENABLE AND CLOCK) THEN INPUT2 --→ STORE --→ OUTPUT
    ELSE                                           STORE --→ OUTPUT
```

c., D-latch with 3-state output

```
    IF NOT OUT_CONT THEN IF INP_EN THEN INPUT --→ STORE --→ OUTPUT
                                   ELSE           STORE --→ OUTPUT
                    ELSE                            Z   --→ OUTPUT
```

Figure 4. Example Actual Device Primitives

```
FUNCTION: MULTIPLEXER WITH 3-STATE BUFFER
FUNCTION 1.
    IF      (ENABLE AND CLK)     THEN INPUT1 --→ STORE --→ OUTPUT1
    ELSE IF (NOT ENABLE AND CLK) THEN INPUT2 --→ STORE --→ OUTPUT1
    ELSE                                         STORE --→ OUTPUT1
FUNCTION 2.
    IF NOT OUTPUT2_CONTROL THEN OUTPUT1 --→ OUTPUT2
                           ELSE    Z    --→ OUTPUT2
```

Figure 5. Example function of a simple module

these attribute values uniquely describe the hologram. Each hologram has a function attribute (key attribute) which will place it into a device class (adder, ALU, processor...). The rest of the attributes are divided into two classes: one, the primary attributes which are associated with the functioning of the device, and second, the secondary attributes express the physical properties of the device. For example, for an adder the primary attributes would be: number of bits, full, output type..., and secondary attributes: propagation delay, fanin, fanout, power supply needed etc.

Each entity class has a restricted attribute set associated with it. This set is large enough that devices can be unique. Also each attribute of this set has a restricted domain which is the set of values an attribute can contain. Typing of the attributes is enforced. The attributes belong to the enumerated data type, and their value is their ordinal value. For each class of entities each attribute is associated with a set of heuristics, from this set a designated one will be used at the evaluation of the similarity of the attribute. The knowledge/data schema for the knowledge base is shown in Figure 6. Because the knowledge base is very large, the uniform handling of knowledge and data is necessary to ensure flexibility of design.

At the comparison of holograms belonging to the same class,

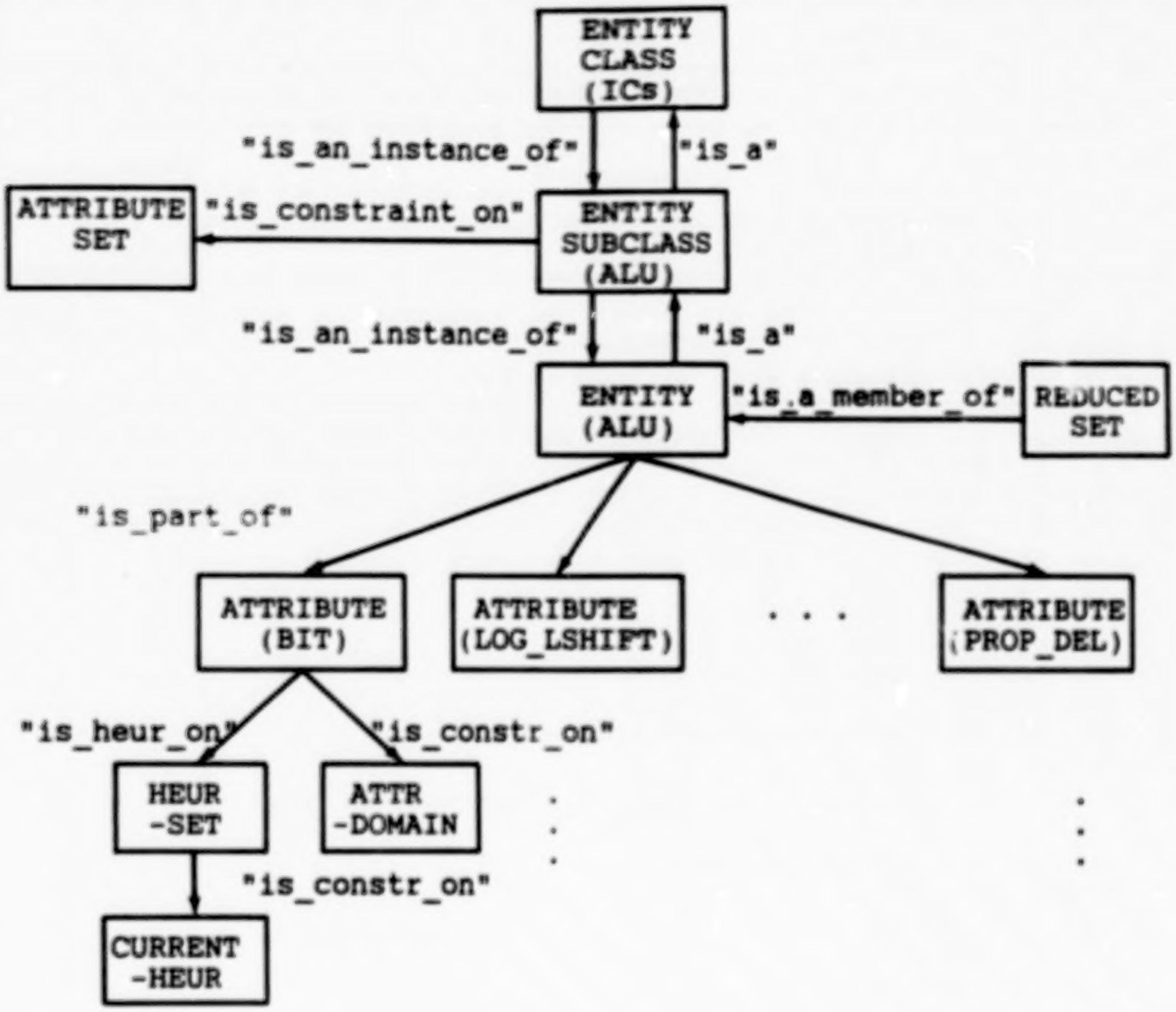


Figure 6. Abbreviated knowledge/data schema

first the primary attributes are processed. Based on the value of the attribute of the candidate device and on the value of the same type of attribute of the device to be matched, and on the associated heuristic, a score is generated. The scores of the primary attributes are added together and a set of devices with the highest score are selected for further processing. The sum of scores for the secondary attributes are generated for this set and added to the sum of primary scores. The device with the highest composite score is selected, and its similarity to the device to be matched is generated comparing its score to the maximum attainable score. Selecting actual devices using attributes and scores is shown in Table 1. Figure 7 depicts a new hologram, which contains a function definition and attributes.

3.3. Name unification

For port name unification there needs to be a way to identify which names correspond to each other in two holograms. Simple positional correspondance can be used in port lists (if there is

ATTR TO MATCH	ADDERS PRI ATTRIBUTES	80		82		83,283		183	
		ATTR	SCORE	ATTR	SCORE	ATTR	SCORE	ATTR	SCORE
FULL	FULL	FULL	1	FULL	1	FULL	1	FULL	1
16	BIT NUMBER	1	0.06	2	0.12	4	0,25	1	0.06
-	NO OF COMP	SINGL		SINGL		SINGL		DUAL	0.06
-	EXTRA FUNC	GATED	0.8	-	1	-	1	CYSAV	0.8
	SUM		1.86		2.12		2.25		1.92

Now devices 83 and 283 are in the set.

ATTR TO MATCH	ADDERS PRI ATTRIBUTES	SN74LS83A		SN7483A		SN74LS283		SN74S283	
		ATTR	SCORE	ATTR	SCORE	ATTR	SCORE	ATTR	SCORE
LS	SERIES	LS	1	-	0.9	LS	1	S	0.8
1	FANIN	1	1	1	1	1	1	1	1
10	FANOUT	5	0.9	5	0.9	5	0.9	5	0.9
1.5	COST	1.6	0.9	1.7	0.8	1.4	1	1.6	0.9
	SUM		6.05		5.85		6.15		5.85

Table 1. Example of selecting devices

```
HOLOGRAM DESCRIPTION: ARITHMETIC LOGIC UNIT
TYPE:  MODULE
KEY ATTRIBUTE: ALU
BIT ATTRIBUTE : 8
SERIES ATTRIBUTE: LS
FUNCTION:
   NUMBER OF SUBFUNCTIONS: 3
   FUNCTION 1:
      FUNCTION : LEFT_SHIFT_ZERO_FILL
      ATTRIBUTES : POSITIVE_EN,POSITIVE_EDGE_TRIG,3_STATE
      IF (S AND CLK) THEN LSZF(A) --→ C
   FUNCTION 2:
      FUNCTION : TWOS_COMP
      ATTRIBUTES : POSITIVE_EN,POSITIVE_EDGE_TRIG
      IF(COMP AND CLK) THEN TCOMP(B) --→ Y
   FUNCTION 3:
      FUNCTION : ADD
      ATTRIBUTES : POSITIVE_EN,3_STATE
      IF COMP THEN ADD(A,Y) --→ C
IN:  A, BUS
     B, BUS
     S, BIT
     CLK, BIT
     COMP, BIT
...
```

Figure 7. A new abbreviated hologram

a convention for port ordering), if the number of ports and their functions match. There is a harder problem if the number of ports and/or their functions do not match. Since each hologram has a function description generated from Actual Device Primitives, the two functions can be compared and names falling in the same position textually can be unified.

4. A METHOD TO DEVELOP A ROOT LEVEL HOLOGRAM

A method to translate a VHDL code to a hologram representation without function definition and attributes has been developed by Klon [4]. To generate the function description of the hologram and the attribute set, an extension of the compiler is necessary. This extension of the compiler will recognize blocks of codes behaving as registers, adders, gates etc. An alternate method is to have device templates as standard components in VHDL which can be filled in at specification definition time. In both cases the attributes will be generated after the hologram function is defined. A method to develop chip level models in VHDL is described in [1].

5. ANALYSIS AND SYNTHESIS PHASE OF THE DESIGN

In the analysis phase the root hologram or specification hologram is decomposed into a tree of modules which have additional connections between them (i.e. signals). The modules correspond to the hierarchy of structural building blocks and the signals are the connections between them.

When a matching hologram is found in the library, it means that the hologram's score attribute is equal to the highest attainable score. Then the hologram and the subtree attached to it are copied with some name unifications. When a similar hologram is found, then the attribute deficiencies are analyzed, and an apropriate action is taken to correct the deficiency. For example if a 16 bit D-latch is needed but the hologram selected only has 4 bits, then four subholograms will be created one for each device. If there is no way to correct the deficiencies, the hologram will be decomposed to its subfunctions and if there are no subfunctions the help of the designer is requested.

Synthesis begins when the root hologram is decomposed, and all leaf nodes are actual devices. Physical device parameters are propagated upward to intermediate holograms until at the root node they are compared to the design requirements. If the design meets the requirements partially, than the source of the problem is, pinpointed and additional requirements are input to the system so as to explore alternative designs.

Once a design is finished , several alternatives can be generated by changing the VHDL code or by adding additional restrictions. To see how a VHDL description can influence the implementation, consider the following. In VHDL several styles of hardware design are supported. These styles are the structural, data-flow, and behavioral styles of descriptions. Additionally in the data-flow design style, it is possible to have control logic and data inextricably woven together; or separate, much like in a state-machine model. These descriptions are equivalent in contextual meaning or semantics. However they imply a different hardware implementation.

6. CONCLUSIONS

In this paper a way to design digital hardware automatically is described. First a VHDL code is developed which is translated to a data structure named hologram. This data structure is a combination of production rules, frames, and semantic networks. Rules facilitate the inclusion of knowledge particular to the hologram, frames help organize heterogenous data, and the semantic networks facilitate the dynamic creation of new designs. The hologram also contains attributes for easy comparison with knowledge base holograms. The decomposition and name unification of holograms is made possible by the function description of the hologram, which is written in a language whose primitives are the functions of actual hardware devices. This hologram is input to the analyzer and synthesizer of the expert system, which after several iterations produces a hardware design. If this design only partially meets the requirements, additional requirements are given and the process is repeated until a design which meets the requirements is generated.

REFERENCES

[1] Armstrong J. R. "Chip Level Modeling with VHDL," E.E. Dept., Va., Tech, Blacksburg, Va., June 1987.

[2] Goering, R. "Silicon Compilers Bridge the Gap between Concepts and Silicon, "Computer Design", November 1987, pp. 67-80.

[3] Green C. R."Development of an Expert Hardware Synthesis System, doctoral dissertation, University of Alabama in Huntsville, Al. December 1985.

[4] Klon, P.F. "On Interfacing HDL to Knowledge Bases", doctoral dissertation, University of Alabama in Huntsville, Alabama, May 1986.

[5] Potter, W. D. and Trueblood, R. P. "Traditional, Semantic, and Hyper-Semantic Approaches to Data Modeling" Computer, June 1988, pp. 53-63.

[6] Shiva, S. G. "Automatic Hardware Synthesis", Proceedings of the IEEE, Vol. 71, No. 1, January 1983, pp.76-78.

[7] Vanderheiden, Robert M. "VHSIC: Midterm Report on a Dynamic Circuit Program, "Defense Electronics, February 1982, pp. 54-62.

[8] "VHDL Language Reference Manual", Draft Standard 1076/B, CAD Language Systems, Inc., Rockville, MD, May 1987.

[9] "VHDL Tutorial for IEEE Standard 1076 VHDL", CAD Language Systems, Inc., Rockville, MD, May 1987.

N89 - 15565

Vulnerability-Attention Analysis for Space-Related Activities

Dan Hays and Sung Yong Lee
Johnson Research Center, University of Alabama in Huntsville
John Wolfsberger, NASA - Marshall Space Flight Center

Abstract. Techniques for representing and analyzing trouble spots in structures and processes are discussed. Identification of vulnerable areas usually depends more on particular and often detailed knowledge than on algorithmic or mathematical procedures. In some cases, machine inference can facilitate the identification. The analysis scheme proposed first establishes the geometry of the process, then marks areas that are conditionally vulnerable. This provides a basis for advice on the kinds of human attention or machine sensing and control that can make the risks tolerable.

Introduction

This paper outlines the main elements of a scheme for analyzing and representing the vulnerability of a structure or process, and for indicating kinds of intelligent attention that could avoid or repair problems at the vulnerable points or regions.

We want to provide tools, mostly computer-based, for depicting trouble spots and for noting their causes and effects. Distinctively, the analysis is explicit about agents, either human or machine-resident, that may be involved in noticing and doing something about potentially harmful situations. The more usual approach is to focus mainly on the devices and processes themselves. However, vulnerability and attention are closely related. Generally, unattended processes are more likely to develop problems, perhaps serious ones. When problems develop in such situations, further problems often ensue. Conversely, attention itself, whether coming from humans or from sensor arrangements, may be open to certain problems such as overload or faulty coordination.

Viewed as an artificial intelligence problem, vulnerability-attention analysis is more a matter of knowledge representation than machine inference, though in handling causal patterns it seems likely that automated techniques could probably save some time. Again, it seems that things go wrong in similar ways for similar structures, so that if structural similarity can be established, then an inference system might suggest looking for certain kinds of problems.

Outline of the Analysis

The steps of the analysis are as follows:

1. Describe the structure or process.
2. Identify vulnerable parts, given specified circumstances.

3. Analyze the causal antecedents of the problems.

4. Trace effects.

5. Based on this analysis, recommend kinds of intelligence and attention that can be applied to avoid or correct the problems.

6. Analyze the allocation of intelligent resources relative to availability, involvement in routine operations, and so on.

There are three domains of information in the analysis:

- the structure or process itself,
- the broader and less well defined realm of factors that could impact the focal structure or process, and
- the sentient resources and their organization.

The structure or process domain—which describes and characterizes the machine, the manufacturing sequence, the managerial procedure—may be the best understood of the three. It is also likely to be somewhat idealized. The second domain, covering various causal factors that can affect the key process or structure, may also be idealized, though conditions of ordinary use are likely to be well enough understood. Objectively, the number of potentially influential external factors is almost always greater than internal sources. Balancing this diversity of possible external causes of problems is the fact that many systems almost always function within a small range of environments. Information about the third domain, that of the sentient resources, is likely to be more variable. If they are humans and only interact occasionally, or if these humans are the designers and testers of the system, they will often be taken for granted, or at least not subject to scrutiny and analysis as part of the system. Generally it seems to be the case that humans who might be involved in detecting and guarding against vulnerabilities will not be considered so systematically as the machines, unless they are involved in operational steps of machine-based processes, in which case they are likely to be treated as components of a mechanized system and paced as such. If the sentient resources are themselves mechanical, they will be treated as part of the physical structure. When the attention of humans is non-routine, for example when some sort of managerial supervision is involved, or when attention is needed only occasionally, as when maintenance or repair is required, the human resources are not likely to be as well planned, since these matters are often not so predictable. In some situations, the human resources may be slighted to no ill effect, when some persons are clever enough to juggle many complex processes. An unjustified reason for skimping on human attention to risky processes is because the accident has not happened yet, or simply to cut costs.

We believe that all humans associated with a system will have an epistemic or knowledge-related role, whether or not this is planned by system designers. They may also have an action role in the process itself. Attempting to understand and represent the dynamics of knowledge in relation to machine and procedure is a major goal of our investigation.

About Vulnerability

Though vulnerability may be thought of as being a property of objects (*intrinsic vulnerability*), it has to do both with the structure or process and with its environment. Trans-situational vulnerability of a system and its parts can be characterized, a sort of "others things equal" vulnerability. But it is usually more informative to describe *contingent vulnerability,* where susceptibility to problems changes with outside factors, history, and so on..

Like much of engineering thought, the ordinary concept of vulnerability focuses on the object (device, process, etc.) rather than on its situations of use. However, it directly implicates situational factors much more clearly than do terms like "risk" or "weakness": a structure is thought of as being vulnerable to something and perhaps as being vulnerable in certain ways.

To say that something or someone is vulnerable means that it may receive effects, ordinarily from external sources. The additional connotation is that the effects may be harmful, or may change the recipient's structure significantly. On examination, practically anything in the universe can be affected by something, resulting either in a change of state or configuration *(ultimate vulnerability)*. . Nevertheless the concept is useful since not everything is equally affected by everything else. Thinking of which parts are relatively vulnerable, or of the overall vulnerability of certain systems, reminds us of some of the associated causal scenarios that could result in changes worth noting, and prepares us to deal with unwanted change.

Students in both engineering and psychology classes were asked to identify the vulnerable aspects of various entities (objects, devices, procedures, people and relationships). They did this easily. In many cases, they linked the undesirable results to a part of the entity, for example, a pump that wears out. In other cases the problematic region was seen more globally, as in a software system that crashed readily but from diverse and unpredictable causes rather than from something more localized such as parameter passing between procedures. Frequently, design problems seem implicated. Students seem to characterize ordinary conditions of use when asked to conceptualize vulnerable aspects of machines. Vulnerable behavior of persons is more often thought of as contingent on unusual circumstances (being away or in a new situation, for example), or as depending on apparently volitional but statistically unlikely actions of the participants. Without going further into discussion of these exercises, the implication is that identifying points of vulnerability seems a natural way to think. Listing the vulnerability of parts is convenient for cataloguing things to watch out for both within a system during operation and in its environment. (Such a list could, of course, obscure causal relations, or lead one to think that the part is somehow responsible for the things that could happen to it involving potent outside sources.)

Representation of Causes and Effects

Even a bare listing of risky areas can be helpful to someone who must deal responsibly with a system. A *surface vulnerability description* can help channel attention and avoid surprises, even when causes of potential problems or remedies for them are not thoroughly understood.

Causal analysis of events or conditions leading to problems will make the depiction more thorough and probably easier to conceptualize.

The depiction of causal paths leading to problems in a structure or process is more of a challenge than just describing the basic system (itself not always an easy task if the system is large or has complex relationships). This is so for several reasons:

- In some cases, more than one causal sequence could lead to the same costly result.
- Causal factors and processes could conceivably be very numerous. Frequently they will reside largely in sources outside the basic system, including ones that are not in the ordinary environment of the system.
- Antecedent circumstances may be described loosely. (Our experience is that level of abstraction problems are likely and pernicious.)
- Causes of problems may not be understood. Thus there may be nothing to represent. (We conjecture that heuristically programmed computer advisory systems might *suggest* problem areas that could be explored.)

One aim of the analysis is to provide graphic representations of causal processes. In doing so, we would like to combine the more abstract tradition of engineering analysis which list causes and effects analytically ("A and B causes event C, which in the case of state D also causes E to happen.") with the kinds of depiction of linked parts or pictures of structures and processes that has been more common in recent computation (for example, in the renderings of semi-animated devices in various AI programs). In vulnerability representations that we have been exploring, causal depictions are linked to parts or regions of a basic schematic or diagram. In some cases, the causal sequences are almost entirely internal to the parts of the focal system, so they can be shown as highlights or certain parameter values of the the basic depiction. In most cases, though, a representation of external events, states, entities, etc. needs to be included, if only as verbal labels. Generally it is the case that alternate causal paths, or simply lists of possible sources and kinds and problems, must be represented for a given combination of system part or region and vulnerability type. Thus, a basic kind of interface with a vulnerability representation would be the familiar one of "selecting" a part or region, or a type of failure (or change), then choosing from a display of alternative possibilities and paths leading to this kind of problem, from a menu, a pop-up, or similar artifice. Because of variations in level of understanding of causal factors, it may not be possible to draw diagrams in all cases. Verbal descriptions are often informative, though for analysis by the system one would probably want more information on connectedness to be included.

A now-classic format for causal depiction is Ishikawa diagrams (see Juran & Gryna, 1980, p. 111), where alternative causal "hypotheses" about failure of parts of systems are attached to arrows which point to parts of an industrial process.

When it comes to showing the *effects* of a problematic state or event, their representation will often be more closely tied to the representation of the focal system, since many of the effects may spread within its structure. However, there may also be various effects on the environment. Some are immediate but side-effects, remote consequences, and other unwieldy contingencies may come about. Generally, the more connected a system is to the outside, either physically or epistemically, the more effects will be representable.

Problems with causal depictions and with causal analysis in general should be noted. Shoham (1988) is one who has recently criticized causal diagrams as being oversimplifications. We feel that they may be useful even if they are something of a simplification of what goes on. Even so, tendencies to oversimplify, to assume that the conceptual space is the real world, to be optimistic about one's favorite devices, to look for simplified causal villains, and so on, must be kept in mind.

Comparisons with Traditional Analysis

Vulnerability-attention analysis does not pretend to compete with traditional analyses of failure or fault. It might be thought of as a representation scheme for some of the material uncovered or formulated under standard methodologies. But some similarities and differences are worth noting.

We think that there is an advantage in representing the actual structure or process in some degree of detail. By contrast, note that probably the most highly developed fault analysis methodology, fault tree analysis (Barlow et al., 1975), depicts Boolean combinations of causal factors and events, rooted in descriptions of major conjectured failure states. Working in a causal space, with a manageable algebra, allows risk coefficients to be computed in a fairly straightforward way, and parts of the causal tree to be scrutinized. It is a strong methodology. The kinds of descriptions urged here, which summarize the geometry of the focal system, together with causal depictions associated with points or regions of the system diagram, are not so neat when it comes to managing them mathematically. However, they have the advantage of calling to mind physical relations of adjacency, which may themselves be causally important. For example, when problems occur owing to accidental connection of parts that are not supposed to be connected (e.g., a solder ball or suffusing gas) it is probably easier to think of these with a process/structure diagram or drawing than with a set of descriptors in an abstract space.

The approach described here is closer in spirit to ordinary failure modes and effects analysis, but would probably lean more toward exploring the connectedness of the entities and events involved more than is sometimes done. (Failure modes analysis is sometimes represented formally; see for example Nielsen, 1975 or Taylor, 1975.)

We would like to be able to reduce some of the labor in identifying cause-effect factors and paths, which is common to all these methods.. One of the questions of vulnerability-attention analysis is whether heuristic analysis, where knowledgeable computer programs interact with subject-matter experts, might reduce some of this labor. It seems that a certain amount of system description, as well as identification of problem-causing paths, depends on particular knowledge of humans. As time goes on it may be possible to incorporate some of this knowledge into computer-based analysis systems or "suggestion systems", in order to reduce some of the tedium of description.

Vulnerability analysis is closer in basic form, though not necessarily in detail, to Ishikawa analysis.

Knowledge Operations

"Attention" is used here roughly as a synonym for "applied intelligence". It serves to point out that knowledge about a system is not useful unless it enters into some real process of noticing, judging, inferring, deciding, adjusting, revising, etc.

Intelligence-in-the-situation requires someone or some knowledgeable machine arrangement.

A variety of persons might be involved with a structure or process at different times. One can distinguish between *pre-attention, on-going attention, and post-attention* relative to an operational phase. Pre-attention consists of efforts to find out possible symptoms of vulnerability, to understand them, and to provide remedies or redesign. On-going attention involves efforts to find symptoms of vulnerability during operations. These might be tipped off by anomalous events, or more directly cued to reliable indicators of specific problems. Knowledge from earlier testing may be useful in this regard. On-going attention may include adjustive or corrective moves. Post-attention evaluates performance, or possibly breakdown, after the fact. Attention at any of these stages could benefit from the knowledge gained at another stage.

These knowledge operations may be *distributed or stratified over agents*. For example, someone who works closely with a machine system will notice small cues that could signal problems. Someone who evaluates statistics of the performance of many such devices may detect more subtle trends. Managers frequently have massive filtering of information, sometimes constrained by regulation or custom, as well as made difficult by communications overload and slippage, that make the evaluation of what is going on quite difficult.

The approach of vulnerability-attention analysis is knowledge-based rather than algorithmic. However, it may use economical and orderly means of identifying causes and tracing effects. Included in the logic of the analysis is to give each part of the focal system a generic identification, so that more general heuristics can be applied to suggest trouble-spots ("This assembly functions as a valve; valves frequently have certain problems; therefore look for....") In other cases, expert and historical case knowledge would be incorporated, since vulnerability of a system and its parts is actually dependent on conditions, and in some cases on particular causal histories.

Risk analysis is often intuitive, certainly knowledge-based, and can be tedious and difficult to represent. Analysis bearing on the best kinds of attention and intelligence is generally less well understood, possibly because people do a good job of it, by and large. Sometimes they do not, however, so more concern with knowledge operations seems to be of utmost importance, especially with costly and high-risk systems, such as may be found in various parts of the Space program.

Background: the Context-Sensitive Scheduling Problem

The analysis discussed in this paper grew out of work which recast a heuristic scheduler for space activities designed by Floyd and Ford (1986) into an object-oriented form (Hays and Davis, 1988). Davis (1988) reprogrammed the Floyd-Ford scheduler, which in its original form used traditional symbolic programming techniques. Though Davis's version maintained an overall flow of control similar to the original, the treatment of discrete processes as "objects" which pass messages to other objects (in this case, scheduling procedures) that evaluate their suitability for location in a schedule, suggested a partially "decentralized" determination of position which was sensitive to power drain, priority, and other factors.

Yet more radically object-oriented approaches to scheduling could be even more suited to nonhierarchically organized environmental contexts. Some higher-level evaluation or conflict resolution is also needed, of course, to prevent local shortsightedness and to insure that a suitable variety of factors are accounted for. This general kind of decentralized "power" situation for computational entities was discussed in Hays (1977).

Other Applications to Space-Related Activities

Since its beginning, work related to space travel and operations has had to consider risky situations. Precise results have had to be obtained in unusual and often dangerous environments. In many cases, the impossibility of operational attention has meant very careful pre-operational attention leading both to rugged, protective designs and to detailed attention during construction, testing, flight preparation, and so on (see for example the discussions in Bolger, 1975).

There are many occasions for representing vulnerability and for understanding the optimal application of knowledgeable attention in space-related activities. Scheduling that is more sensitive to context is just one. Testing of devices and procedures, ordinary management for development, design of operational environments with shared machine and human monitoring and decision-evaluation, and various other computer-assisted operations, are all candidates for vulnerability-attention analysis. Deeper understandings of knowledge operations, and the relation of knowledge to external process, should produce efficiencies of analysis and of performance.

Acknowledgements

The work upon which this paper is based has been supported in part by Grant #NAG8-641 from Marshall Space Flight Center, Donnie Ford, Principal Investigator, John Wolfsberger, Project Monitor. Please address correspondence to Dan Hays, 135 Morton Hall, University of Alabama in Huntsville, Huntsville, AL 35899.

References

Barlow, Richard E., Fussell, Jerry B. & Singpurwalla, Nozer D. (Eds.). *Reliability and Fault Tree Analysis.* Philadelphia: Society for Industrial and Applied Mathematics, 1975.

Bolger, Philip H. (Ed.). *Space Rescue and Safety 1975.* Vol. 41, Science and Technology. American Astronautical Society, 1975.

Floyd, S. & Ford, D. in *Proceedings of the Conference on Artificial Intelligence for Space Applications,* Huntsville, 1986.

Hays, D. "Dominance relations in computing systems." AFIPS Press: *Proceedings of the 1977 National Computer Conference,* 1977, pp. 595-600.

Hays, D. Davis, S., and Wolfsberger, J. "Implementation of a scheduler in LISP and in Ada." *Robotics and Automation Conference,* Huntsville, 1988.

Juran, Joseph M. & Gryna, Frank M. Jr. *Quality Planning and Analysis: from Product Development through Use.* Second Edition. New York: McGraw-Hill, 1980.

Nielsen, Dan. "Use of Cause-Consequence Charts in Practical Systems Analysis." In Barlow, et al., 1975, pp. 849-880.

Shoham, Yoav. *Reasoning about Change.* Cambridge, MA: MIT Press, 1988

Taylor, J. R. "Sequential Effects in Failure Mode Analysis." In Barlow, et al., 1975, pp. 881-894.

N89-15566

GRAPH–BASED REAL–TIME FAULT DIAGNOSTICS

S. Padalkar, G.Karsai and J. Sztipanovits
Vanderbilt University, Nashville, TN 37235 USA

ABSTRACT

A real-time fault detection and diagnosis capability is absolutely crucial in the design of large-scale space systems. Some of the existing AI-based fault diagnostic techniques like expert systems and qualitative modelling are frequently ill-suited for this purpose. Expert systems are often inadequately structured, difficult to validate and suffer from knowledge acquisition bottlenecks. Qualitative modelling techniques sometimes generate a large number of failure source alternatives, thus hampering speedy diagnosis.

In this paper we present a graph-based technique which is well suited for real-time fault diagnosis, structured knowledge representation and acquisition and testing & validation. A Hierarchical Fault Model of the system to be diagnosed is developed. At each level of hierarchy, there exist fault propagation digraphs denoting causal relations between failure-modes of subsystems. The edges of such a digraph are weighted with fault propagation probabilities and fault propagation time intervals. Efficient and restartable graph algorithms are used for on-line speedy identification of failure source components.

INTRODUCTION

A very high degree of automation and complexity is evident in modern industrial plants and space systems. This trend towards fully automatic and largely unmanned complex systems necessitates the development of a real-time fault detection and diagnosis capability. Such a capability would lead to shorter repair times and longer system operational times, thus enhancing productivity. Signal processing techniques coupled with modern sensors are capable of fault detection and alarm generation. Advances in computer technology such as multi-processing allow improvements in real-time performance. New artificial intelligence (AI) programming techniques, such as declarative languages and symbolic processing are very efficient for representing and processing the failure models of systems.

PROBLEM STATEMENT

A real-time fault diagnostics system has to function in an environment where new alarms may constantly be generated, due to the propagation of failures. To cope with such a time-changing scenario the diagnostics system must have the following characteristics:

- Signal Processing, Alarm Generation and Failure Source Identification software must be as fast as possible. The first two are usually standard well-defined and analyzed algorithms, and hence, virtually all speed improvements have to be achieved in the failure source identification phase.
- The diagnosed results must be updated as time elapses and new alarm information is received. These results must be accurate but need not have a fine resolution. This implies that in the early stages of diagnosis a large

component such as the Gas-Delivery Assembly can be identified as the fault source. The resolution of this fault source is further refined with the passage of time and additional alarm information to a unique valve inside the Gas-Delivery Assembly.

- The User-Interface must present the current status of diagnosis in a comprehendible manner, reflecting the level and the granularity of the system under diagnosis, at which the diagnostics system is operating.

SURVEY OF OTHER TECHNIQUES

Rule-Based approaches or Expert Systems (1) have been the primary AI technique used for fault diagnostics. Expert Systems use IF - THEN rules as their knowledge representation structure, and an inference engine operating on these rules for detecting the source of failure. For diagnosing large-scale systems, Expert Systems are often unsuitable since they cannot be efficiently modularized. Large Expert Systems also suffer from maintanence, testing and validation problems. Often large Expert Systems remain incomplete because of knowledge acquisition problems.

Another approach has centered on using qualitative models (2) which are a simulation of faulty system behavior. Fault sources are identified by comparing incoming data with all possible qualitative simulation models until a match is found. This process may generate too many models and be too time-intensive for use in real-time fault diagnostics.

A variety of graph-based techniques such as fault trees (3), event-trees (4) and cause-consequence diagrams (5) have also been used for fault diagnostics. An interesting approach proposed by Narayanan and Vishwanadham combines hierarchical fault propagation digraphs with fault trees (6). It is judged that a graph-based technique offers the best hope for real-time fault diagnostics.

GRAPH-BASED APPROACH

The basic philosophy of our graph-based approach is based upon Multiple-Aspect modelling. The system under consideration is hierarchically decomposed from many aspects in order to yield many different models. A functional decomposition leads to the Hierarchical Process Model (HPM) and a structural decomposition leads to a Hierarchical Physical Component Model (HPCM). A Hierarchical Fault Model (HFM) is developed in the context of HPM with links to the HPCM.

The technique of hierarchical decomposition is widely used during model building for the following reasons:

1. Design, knowledge acquisition and knowledge-base maintanence of large complex system becomes structured and easier.

2. Running the same graph algorithms on smaller number of nodes many times takes lesser time than running them on the entire set of nodes in a system. For example it takes longer time to run an $O(n^3)$ algorithm on a graph with 200,000 nodes than it takes to run the same algorithm 200 times on a graph with 100 nodes.

3. It is possible to conduct the search for the failure source on the HFM in a parallel manner, thus enabling speedy diagnosis.

4. In most cases a large granularity component assembly can be identified as a failure source at an early stage, and then the search need only proceed in that component's part of the model.

Hierarchical Process Model

A process in the HPM can be thought of as a functional unit carrying out a specific function in the system, by utilizing different physical components. Different processes on the same level may interact with each other through shared physical components. Processes in the HPM can be associated with many different components in the HPCM as shown in figure 1. In the context of each process the following are acquired:

1. Process Failure-Modes.
2. Process Alarms and alarm-generators. The alarm-generators accept sensor inputs and if needed, generate the appropriate alarm.
3. Alarm Failure-Mode associations.
4. Failure-Mode Physical Component associations.

Hierarchical Fault Model

Each process in the the HPM also has its fault model. This model is derived from that its failure-modes, and if present, the failure-modes of its subprocesses. All these failure-modes form nodes of a fault propagation digraph, with directed edges between individual failure-modes signifying a fault propagation possibility. Each edge in this graph is weighted with two parameters a fault propagation probability and a fault propagation time interval in terms of a minimum and a maximum. The fault propagation digraph of a process on level i is shown in figure 2. The collection of all such fault propagation digraphs and failure-mode physical components associations results in the HFM. It is possible to extract a rough fault propagation digraph from process physical interactions since most faults can only propagate along physical connections.

Model Building

The process of model building and specification was aided by graphical editing tools developed at Vanderbilt University (7). Each model was built using its its own specific editor and also had its own declarative specification language. The output of an editor is the specification of a model in its declarative language. A sample output of the fault model editor is shown in figure 3. These generated specifications are used by special-purpose interpreters for generating the run-time environment of the real-time fault diagnostic system.

DIAGNOSTIC ALGORITHMS

By running suitable algorithms on a fault propagation digraph, a failure source process and its source failure-mode can be found. Since each failure-mode in each process is also associated with physical components, the source faulty components can also be found. This process can be migrated to lower levels of process hierarchy in order to get a better resolution. Hence the failure source identification process consists of two algorithms the Failure Source Process Identification (FSPI) and the Fault Source Component Identification (FSCI). An Inter Level Migration (ILM) process does the task of searching the process hierarchy for the best resolution of the possible faulty source component.

Failure Source Process Identification

The FSPI algorithm gets as input the fault-propagation digraph of a process to be diagnosed. It also receives all alarms currently ringing within that process and its subprocesses. This algorithm is accurately capable of detecting under most circumstances, whether a single or a multiple fault occured in the process. On completion, this algorithm returns the possible fault source subprocesses and their fault source failure-modes. It uses the following constraints to determine the fault source in case of a single fault condition :

1. Reachability Constraint : All ringing alarms shall be reachable from the detected source failure-modes.
2. Monitor Constraint : No failure-mode with a normal alarm shall lie on a path from any of the detected source failure-modes to any of the failure-modes with a ringing alarm.

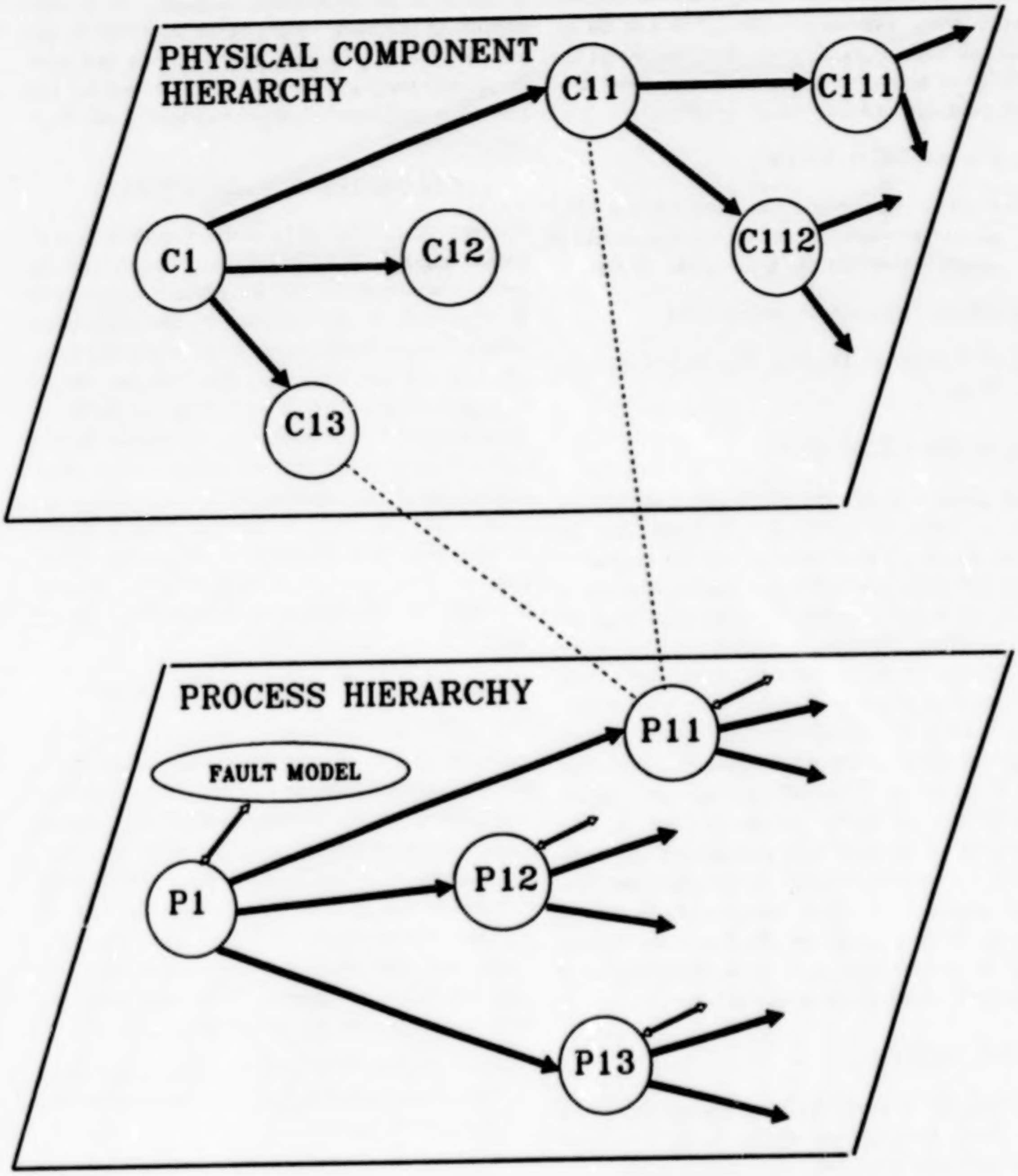
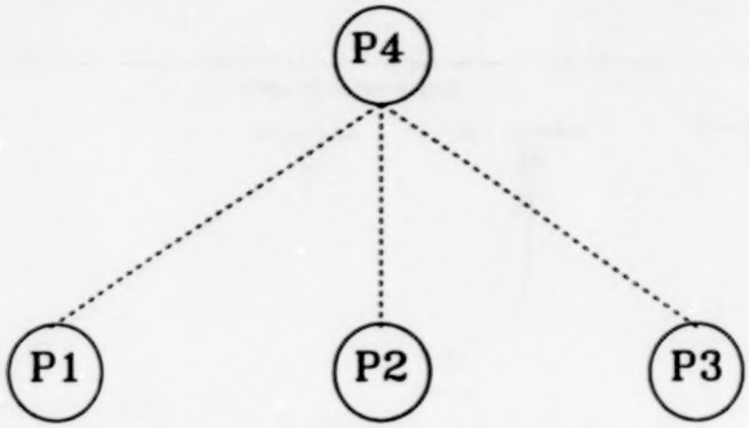


FIGURE 1 Multiple–Aspect Modelling

Process Structure On Level i :



Fault Model On Level i :

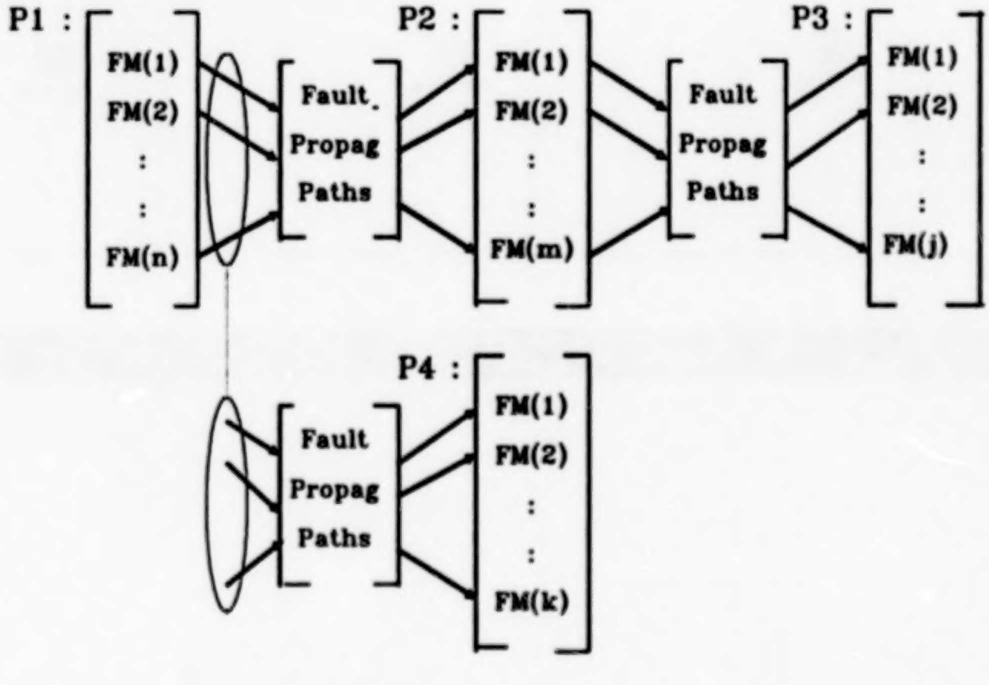


FIGURE 2: Fault Propagation Digraph

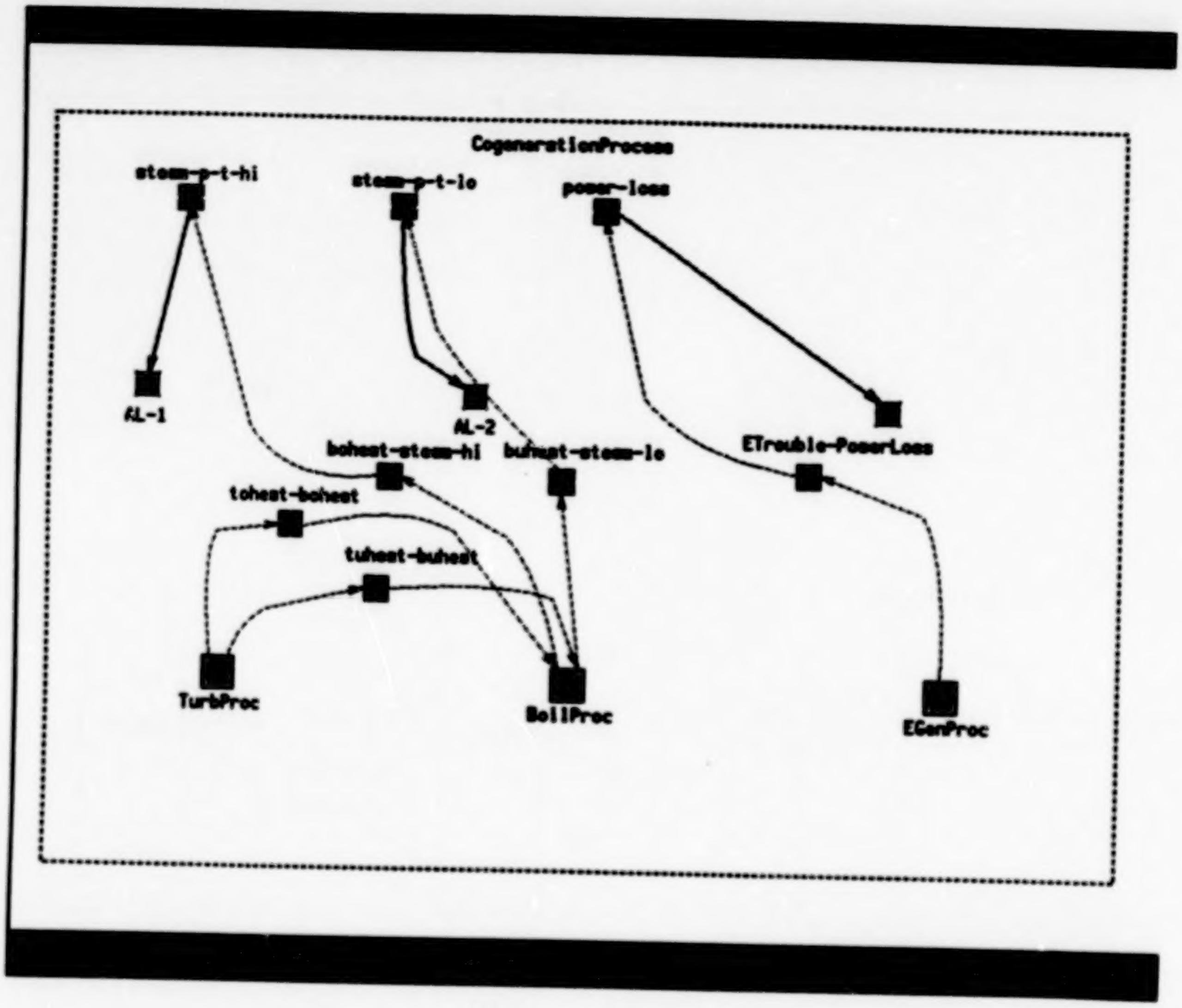


FIGURE 3: Fault Model Editor Session

3. Temporal Constraint : All ringing alarms shall be individually reachable from the detected individual source failure-modes within the time interval computed from the time intervals found on shortest path between each individual alarm and source failure-mode pair.

4. Consistency Constraint : There shall be no failure-mode with a ringing alarm whose reachability time from a source failure-mode is greater than, the maximum reachability time of a failure-mode with a normal alarm from that detected source failure-mode.

The algorithm is closed and complete and is thus suitable for speedy location of failure source processes.

Fault Source Component Identification

The FSCI algorithm takes as input a list of detected source failure processes and their source failure-modes. In case of a single fault condition it returns a union of all physical components associated with the source failure-modes. In case of a multiple fault condition it tries to find a common component amongst all the source failure-modes. If successfull it returns that common component, and if not it returns a union of all associated components.

Inter Level Migration

The ILM process detects the highest level of the process in which alarms are ringing. It then tries to search for a failure source by running the FSPI and later the FSCI algorithms on all processes in that level. The results are used to guide a breadth-first search of all processes present in the next lower level. This process continues until the lowest level of hierarchy is reached. At this point the best possible resolution of the failure source is achieved. If during this migration an alarm rings in a higher level than the current one under processing, the ILM goes to that higher level and restarts the diagnosis. At any point in time the ILM can present its best guess of the failure source in any level of process hierarchy.

DIAGNOSTIC SYSTEM ARCHITECTURE

The Real-Time Fault Diagnostic System required:

1. The use of a distributed computing architecture,
2. Support for a concurrent programming model and
3. Integration of symbolic and numerical computations.

The Multigraph Architecture (MGK) (8) has been used as a generic framework in the implementation of the diagnostic system. The MGK is dataflow oriented computing system, capable of allocating computing nodes on a distributed network consisting of uniproccessor as well as multiprocessor configurations. The language of the these computing nodes can be Lisp or C or Ada, thus enabling integration of symbolic and numerical computations. The MGK supports programming models such as autonomous communicating objects (9).

The diagnostic system architecture is shown in figure 4. A Monitor task handles the job of acquiring sensor outputs and alarm-generation. The Diagnostic task consists of a diagnostic manager object, a diagnostic methods object and a display manager object. The diagnostic manager accepts as input all generated alarms and is in charge of conducting the inter-level search for the failure source. During this search it may send a process to the diagnostic methods object asking it to perform either the FSPI algorithm or the FSCI algorithm on it. The diagnostic methods object performs the requisite algorithm and reports the result back to the diagnostic manager. These results are used by the diagnostic manager

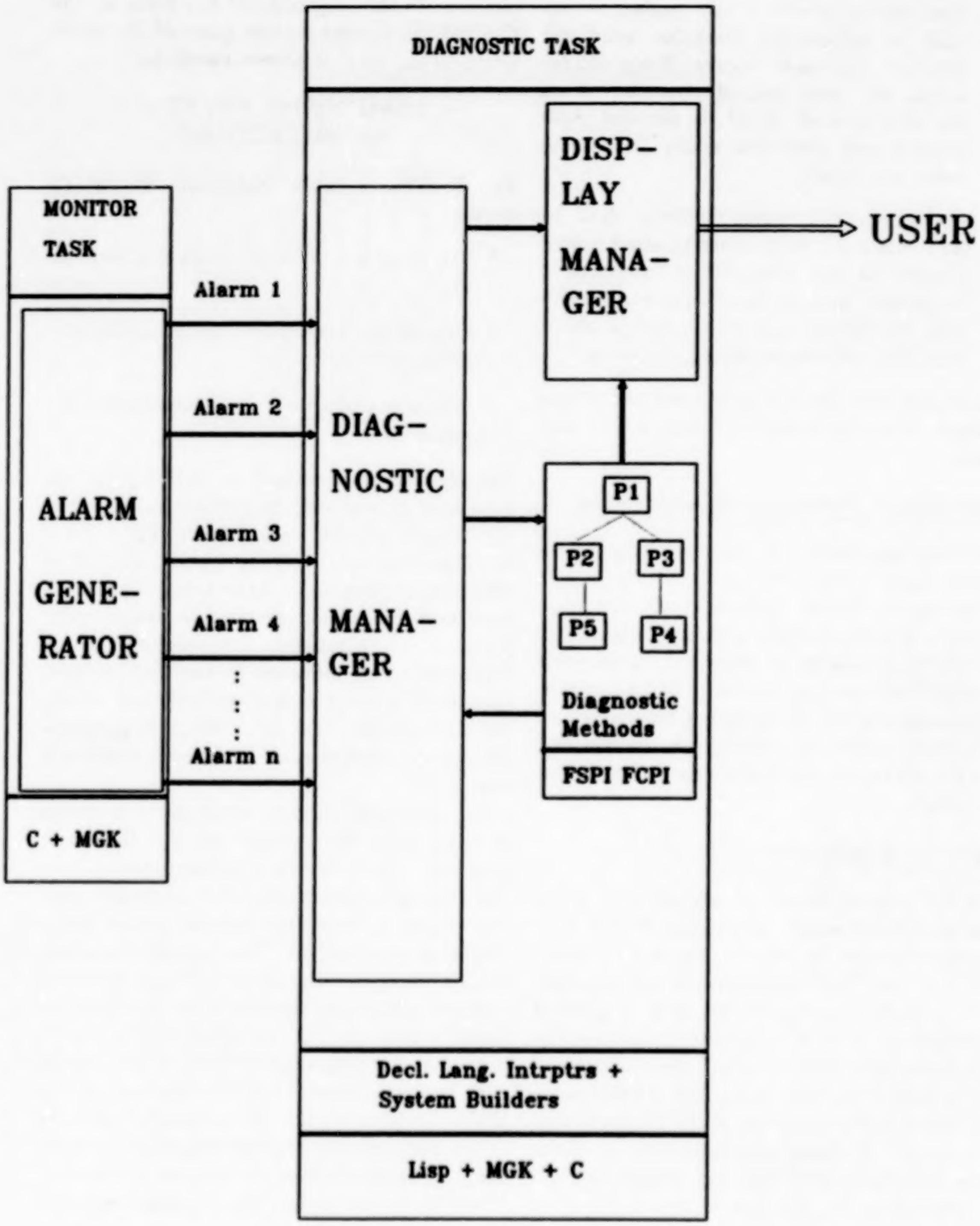


FIGURE 4: Diagnostic System Architecture

as a guide in its search. As soon as results are obtained for a level in the hierarchy they are sent over to the display manager for displaying them to the user.

TESTING AND VALIDATION

A real-time alarm pattern simulator is used to test and validate the real-time fault diagnostics. This simulator is automatically derived from the HFM. This simulator accepts as input any number of failed components and the times at which they are supposed to have failed. It then generates in real-time the pattern of alarms that would ring due to the failed components. These alarms serve as the input for the diagnostics system.

CURRENT STATUS

A real-time fault diagnostics system for a Cogenerator plant currently exists on an HP 9000/300 computer. The process model has 5 levels of hierarchy and 45 processes. The average number of failure-modes per process is about 4 and hence the average number of nodes in a fault propagation digraph is about 10. An alarm pattern simulator is currently being used to test and validate the diagnostics system. A test in which 5 alarms were generated in a span of 20 seconds, the diagnostics system located the failure source with the finest possible resolution in 25 seconds.

CONCLUSIONS

A hierarchical graph-based model appears to be the most suitable fault model for real-time fault diagnostics. The knowledge acquisition process can be automatized to a large extent. The graph algorithms developed are fast enough to be used for speedy diagnosis. The system is able to update itself and restart the diagnostic procedure if necessary. The breadth-first search strategy enables the system to provide an accurate diagnosis of a coarse resolution that can be refined with passage of time and more alarm information. Testing and validation of such a system is easier since the test program can be automatically generated from the fault model itself.

REFERENCES

(1)Laffey, T., Perkins, W., Nguyen, T., "Reasoning about Fault Diagnosis with LES", First Conference on A.I. Applications, Dec. 1984.

(2)Milne, R., "Strategies for Diagnosis", IEEE Transactions on Systems, Man, and Cybernetics, May/June 1987.

(3)McCormick, N., "Reliability and Risk Analysis", Academic Press Inc. New York, New York, 1981.

(4)Kumagai, N., Ishida, Y. and Tokumaru, H., "A Knowledge Representation for Diagnosis of Dynamical Systems", IFAC Real Time Programming, Lake Balaton, Hungary 1986.

(5)Himmelblau, D., "Fault Detection and Diagnosis in Chemical and Petrochemical Processes", Elsevier Scientific Publishing Company, Amsterdam, Netherlands, 1978.

(6)Narayanan, N. and Vishwanadham, N., "A Methodology for Knowledge Acquisition and Reasoning in Failure Analysis of Systems", IEEE Transactions on Systems, Man, and Cybernetics, March/April 1987.

(7)Karsai, G., "Graphical Description Package", unpublished memorandum, Dept. of Electrical Engineering, Vanderbilt University, 1986.

(8)Sztipanovits, J., "Execution Environment for Intelligent Real-Time Systems", Proc. of the NASA Workshop on Telerobotics, Pasadena, CA, 1987.

(9)Sztipanovits, J., Biegl, C., Karsai, G., Padalkar, S., and Purves, R., "Programming Model for Coupled Intelligent Systems in Distributed Execution Environments", Proc. of SPIE's symposium on "Advances in Intelligent Robotic Systems", Cambridge, MA, 1886.

N89 - 15567

AUTOMATIC DETECTION OF ELECTRIC POWER TROUBLES

(ADEPT)

Caroline Wang, Hugh Zeanah, Audie Anderson, Clint Patrick
Information and Electronic Systems Laboratory
Marshall Space Flight Center, NASA
Huntsville, Alabama

Mike Brady and Donnie Ford
University of Alabama in Huntsville

ABSTRACT

ADEPT is an expert system that integrates knowledge from three different suppliers to offer an advanced fault-detection system, and is designed for two modes of operation: real-time fault isolation and simulated modeling

Real time fault isolation of components is accomplished on a power system breadboard through the Fault Isolation Expert System (FIES II) interface with a rule system developed in-house. Faults are quickly detected and displayed and the rules and chain of reasoning optionally provided on a Laser printer.

This system consists of a simulated Space Station power module using direct-current power supplies for Solar arrays on three power busses. For tests of the system's ability to locate faults inserted via switches, loads are configured by an INTEL microcomputer and the Symbolics artificial intelligence development system. As these loads are resistive in nature, Ohm's Law is used as the basis for rules by which faults are located.

The three-bus system can correct faults automatically where there is a surplus of power available on any of the three busses. Techniques developed and used can be applied readily to other control systems requiring rapid intelligent decisions.

Simulated modeling, used for theoretical studies, is implemented using a modified version of Kennedy Space Center's KATE (Knowledge-Based Automatic Test Equipment), FIES II windowing, and an ADEPT knowledge base. A load scheduler and a fault recovery system are currently under development to support both modes of operation.

PRECEDING PAGE BLANK NOT FILMED

INTRODUCTION

Marshall Space Flight Center(MSFC) is involved in design and development of the Automation of Electrical Power Systems project. This demonstrates the feasibility of using computer software to enhance fault-diagnosis techniques and develop fault-recovery techniques for the Space Station. To accomplish this, prototype software was developed to automate such tasks as detecting and isolating faults and monitoring and reasoning status.

The ADEPT system includes:

(1). Real time fault isolation through a breadboard modeling the power components.

(2). A local simulator which uses the theoretical models and will eventually support the fault recovery system.

HISTORY BACKGROUND

In 1985, Martin Marietta Denver Aerospace delivered to MSFC the Fault Isolation Expert System including a two-rack, 350-watt, three-channel electrical power system breadboard.

MSFC was experimenting with various software techniques to improve performance and speed. ADEPT was built with the MSFC rule system utilizing the existing FIES II breadboard and software interface and KATE as a tool for the local simulator.

The real-time fault isolation version was implemented in LISP, because of its ability to search for a fault, display fault data, and automatically print out the fault reasons and current data along with the steady-state data for comparison.

The University of Alabama in Huntsville is also involved in this project. They have already converted the software into Symbolics system genera 7.1, and also will be conducting a future study of load management and scheduling.

THE ADEPT SYSTEM

ADEPT is composed of a Symbolics 3670 computer linked to the modified FIES II system. The Symbolics 3670 includes a high-resolution graphics terminal, eight megabytes of memory, a 474 megabyte hard disk, Laser graphics printer, and a LISP environment. The FIES II breadboard is built into two side-by-side racks containing the host computer, its memory storage devices and I/O support equipment; the relay board subrack, its power supplies and related controllers; communications boards, ports, and cables; housekeeping power supplies; control switches and lighted displays.

Figure 4 outlines the components in the ADEPT system and the interactions of these components with one another.

Data transfer scheduling and control are provided by the host computer, an Intel System 86/380. Based on the iRMX86 operating system, the 86/380 contains the iSBC 86/30 Single Board Computer board, a thirty-five megabyte Wincester hard-disk, a one megabyte eight-inch flexible disk drive, and a multibus expansion rack with slots containing not only controllers for the computer itself, but also communication and data conversion boards discussed below. Software run on the 86/380 is written in Intel's ASM86 assembly language.

Three dual-sided power supplies provide charge to the batteries or electricity to drive the system's load resistances, or both, depending upon the configuration into which the busses' relays are set. Representing the Space Station's solar arrays, these supplies are capable of up to fifty volts and nearly two amperes output on each of the six available channels. Each supply has independent current-limiting adjustment, allowing simulation of various solar array lighting conditions.

At the heart of the breadboard is the relay board subrack, comprised of six boards containing forty-eight relays along with related support components. In addition to its function as the system's switching center, the relay subrack provides attach points for most of the sensor lines, by which A/D converters sample system conditions, and all of the fault insertion lines. The fault insertion logic, used to introduce various abnormalities at nodes along the power busses, sends its outputs directly to the relay boards where the "support components" mentioned effect conditions of open or closed relay faults and resistive or direct shunt faults. Configurations may be inserted either manually, using toggle switches on the front panel of the FIES II racks, or remotely, from the Symbolics terminal or the debugging monitor. In the event that a switch is offset from the normal mode, the corresponding relay cannot be controlled remotely, but a fault will exist and should be detected and isolated.

The system integrates software from three different suppliers to offer an advanced fault detecton system, designed for the two modes of operation outlined in Figure 1.

Real-time fault isolation of components on the power system breadboard through the FIES II interface is made possible with the MSFC rule system. Faults are quickly detected, displayed, and reasoning provided. The FIES II interface and MSFC rules system were written in Common Lisp (Figure 2).

The simulation version uses a frame-based system, describing all system components and the relationships among them. A constraint system analyses these relationships and compares theoretical to actual measured values, thus identifying constraint failures(Figure 3).

FAULT ISOLATION

When an initial configuration of loads is selected and downloaded from the Host Computer, and steady-state condition is achieved, all the sensor points' voltages and currents are read continuously and any significant change at any sensor point indicates a fault has been inserted. This Fault condition is then flagged to the Host computer to initiate the isolation program.

Fault Type: OPEN RELAY

Open circuit conditions are indicated by a sudden drop in the values of current read at the sensors while the voltage values remain the same or perhaps higher. Isolation is done by searching for a sensor point where the voltage is zero. The location of the inserted fault lies between the sensor points where there was a voltage and was not a voltage.

Fault Type: DIRECT SHUNT

A direct shunt, or short-circuit, fault causes a sudden increase in the sensor readings of current values and a decrease in voltage values on sensors nearest the power source. When this occurs, the fault type is identified and a search begins for a sensor point where the current is higher than the steady state current and following points have current readings of approximately zero.

Fault Type: RESISTIVE SHUNT

Resistive shunt causes a sudden increase in current readings on the sensors nearest the power source. A decrease in the voltage may also occur where the load plus the resistive shunt causes the current to exceed the capacity of the solar cells being simulated. Isolation of the resistor shunt fault is done by identifying the first sensor reading with a significant decrease in current. The fault is between this sensor and the last one back toward the power source with a high current reading.

REFINEMENTS

Refinements in the rules are made using Ohm's Law to further identify the type of fault being experienced. This is done by considering the ratio of the values of currents and voltages between steady-state and fault conditions.

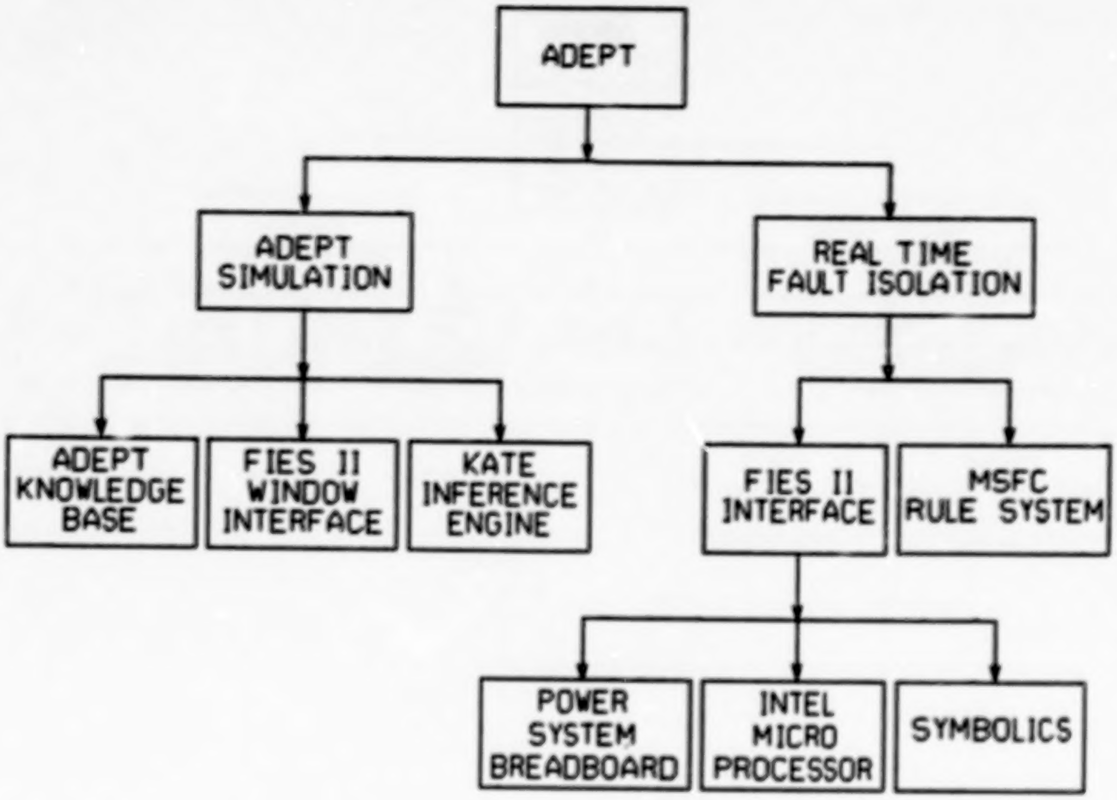


Figure 1. ADEPT system flow diagram

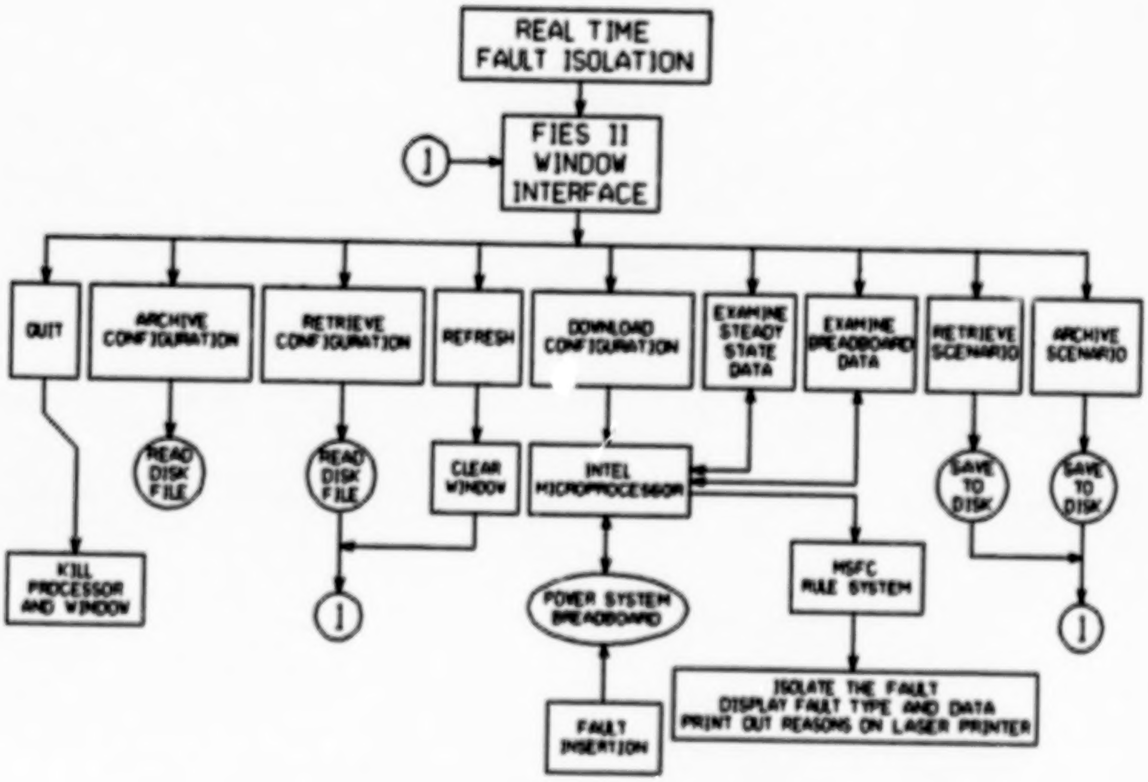


Figure 2. ADEPT real-time fault isolation flow diagram

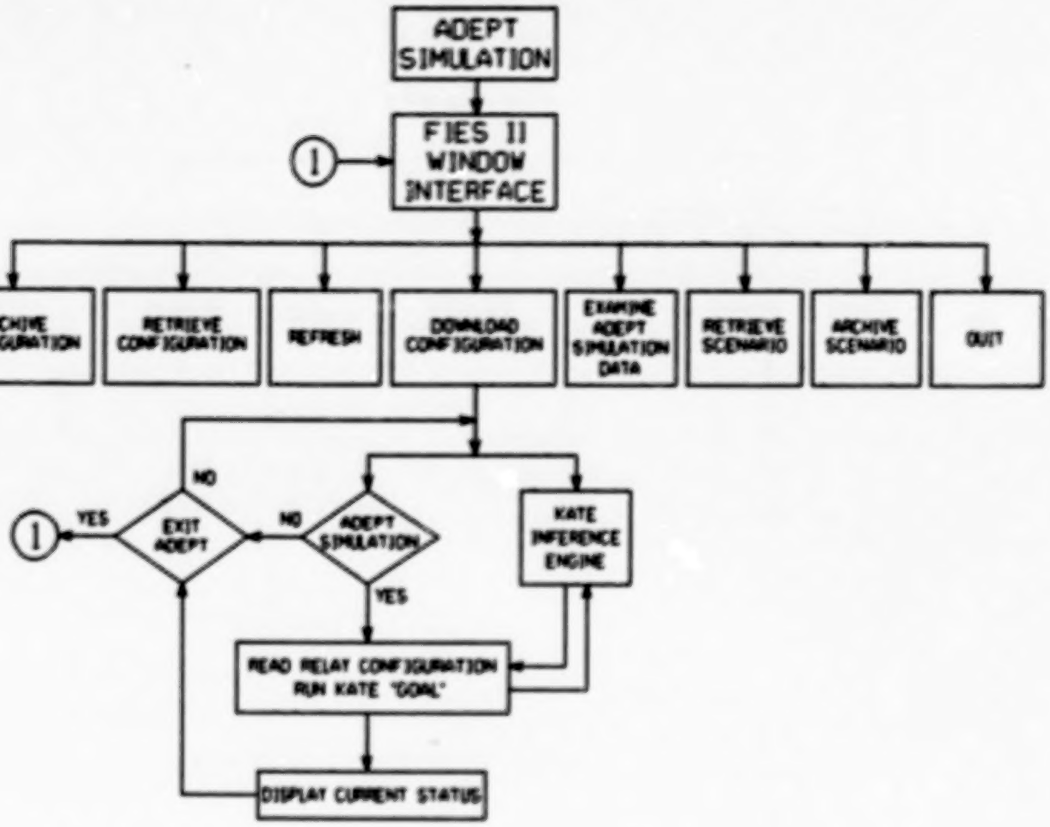


Figure 3. ADEPT simulation flow diagram

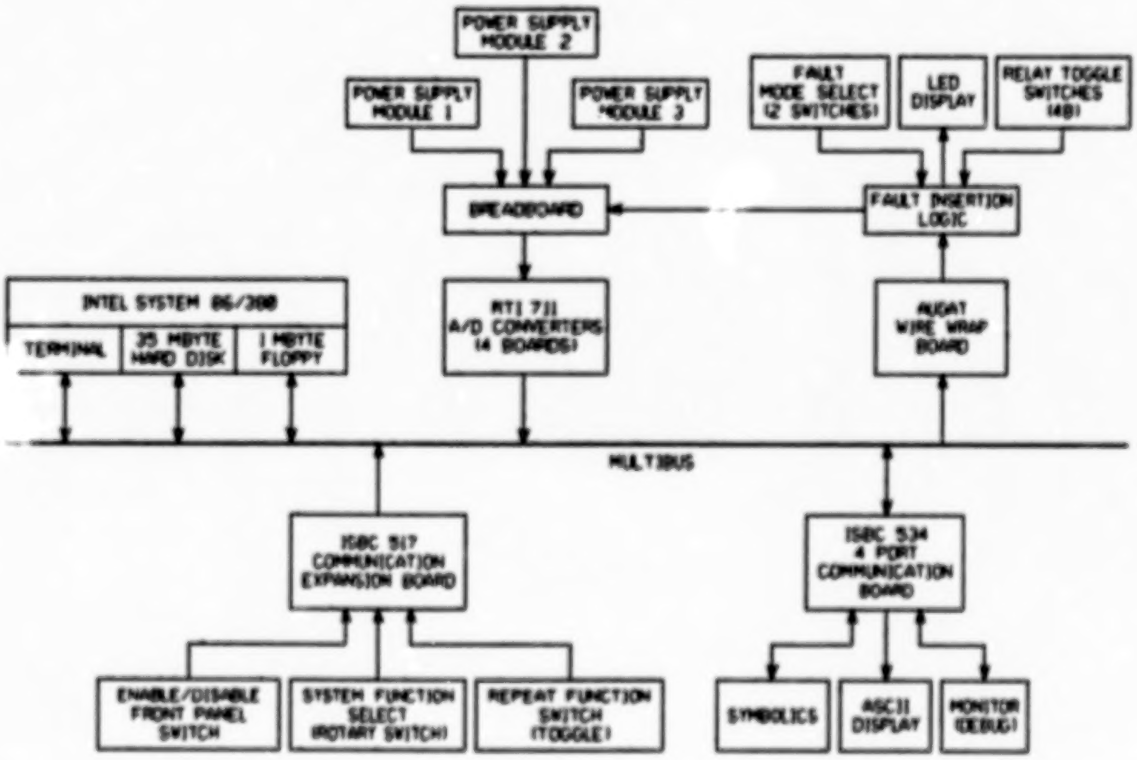

POWER SUPPLY MODULE 2
POWER SUPPLY MODULE 1
POWER SUPPLY MODULE 3
FAULT MODE SELECT (2 SWITCHES)
LED DISPLAY
RELAY TOGGLE SWITCHES (48)
BREADBOARD
FAULT INSERTION LOGIC
INTEL SYSTEM 86/380
TERMINAL
35 MBYTE HARD DISK
1 MBYTE FLOPPY
RTI 711 A/D CONVERTERS (4 BOARDS)
AUGAT WIRE WRAP BOARD
MULTIBUS
ISBC 517 COMMUNICATION EXPANSION BOARD
ISBC 534 4 PORT COMMUNICATION BOARD
ENABLE/DISABLE FRONT PANEL SWITCH
SYSTEM FUNCTION SELECT (ROTARY SWITCH)
REPEAT FUNCTION SWITCH (TOGGLE)
SYMBOLICS
ASCII DISPLAY
MONITOR (DEBUG)

Figure 4. Hardware flow diagram

N89-15568

AN OVERVIEW OF VERY HIGH LEVEL SOFTWARE DESIGN METHODS

Maryam Asdjodi and James W. Hooper
Computer Science Department
The University of Alabama in Huntsville
Huntsville, Alabama, 35899

ABSTRACT

Very High Level design methods emphasize automatic transfer of requirements to formal design specifications, and/or may concentrate on automatic transformation of formal design specifications that include some semantic information of the system into machine executable form.

Very high level design methods range from general domain independent methods to approaches implementable for specific applications or domains. Applying AI techniques, abstract programming methods, domain heuristics, software engineering tools, library-based programming and other methods different approaches for higher level software design are being developed. Though one finds that a given approach does not always fall exactly in any specific class, this paper provides a classification for very high level design methods including examples for each class. These methods are analyzed and compared based on their basic approaches, strengths and feasibility for future expansion toward automatic development of software systems.

INTRODUCTION

Automatic programming is one of the long range goals of computer science research. Understanding the natural language interface, converting the specifications in natural language to formal design specifications, and developing implementations are constituent components of automatic programming [2]. Natural language understanding has been an evolutionary process. In its actual implementation, automatic programming always is viewed as substitution of a higher level language for specifying a system to a machine for the languages that are presently available [14]. In order to avoid ambiguity and make the problem manageable, a limited set of vocabulary and interpretation rules are used for the machine interface. Compilers are among the primary tools that improved software specification and introduced basic generic and reusable programming concepts (e.g., loop structures). They allowed higher level specifications than what a machine by its nature was designed to understand. Specification of a software system in a high level language, should be based on specific syntactic rules (BNF) of the language. Compilers are designed to verify software specification (i.e. program) correctness by detecting mainly the syntactic errors of the implementation and to develop executable specifications (i.e. machine code) for correct programs. Syntactic errors are not the only inaccuracy of programs. Logical and semantic errors result in a larger class of faulty programs. Semantic information includes the definition of objects, relations, rules, and algorithmic concepts that are used for describing the system. Errors related to these interpretations usually are referred to as semantic errors [13]. Semantic errors result from misrepresentation or misunderstanding of the meaning of the requirements or design parameters. Compilers for high level languages such as FORTRAN detect few of these errors.

Very high level (VHL) design methods are being developed by moving up toward greater abstraction of specifications and automatic software generation by relaxing syntactic rules of high level languages, and/or including more semantic information in design specifications. The designer's knowledge about the real system is represented by different methods. Object-oriented programming incorporates a view of real-life entities in terms of their functions and relations with other entities. Logic-based programming models a system in terms of logical statements and assertions. Application of artificial intelligence methods for designing software systems is recommended for use by software engineers [17, 20]. Transformation techniques are used for converting VHL design specifications into implementations. Knowledge-based systems are used for defining an application domain to a computer. What is common in all of these approaches is the necessity of more generic and adaptable constructs for VHL specification of a software system. These reusable aspects of VHL design tools range from standard methodology and control structures of design to generic objects and library components. The next section provides a review of VHL design methods and their approach to reusability.

CLASSIFICATION OF APPLICATION

The design of a software system refers to specification of its algorithmic concepts, data structures, functional components, and interfaces between these components [12]. It is the most important and crucial phase of the software life cycle. Adaptable and more abstract designs, when automatically transformed to implementation in high level languages, release the software system designer from dealing directly with the syntax of programming languages, resulting in more reliable implementations. Different VHL design approaches emphasize reusability of specification, structure, and methodology of the software design, in a different level. They range from efforts to develop generalized structural design methods for transforming informal requirements of problems to formal design specification, to approaches for implementing predefined design elements. Although VHL design approaches are very diverse, they are grouped in the following categories with respect to their major approaches.

- General approaches.
  - Software engineering approaches.
  - Program transformation approach.
  - Component composition approach.
- Application-oriented Methods.
  - Knowledge-based approach.
  - Application language approach.
  - Object-oriented programming.

GENERAL APPROACHES

General VHL design methods provide means for designing a system by applying design languages, environments and tools that are independent of the application domain. General VHL design methods allow more validation of specification of designs by implementing general programming and software design knowledge for developing VHL specifications and transforming them to software. In most cases logical, functional, or relational design approaches are enforced by general VHL design methods. Generally these systems are interactive and no knowledge of any application field is required. The following subsections describe classes of approaches in this category.

Software Engineering Approaches

Software engineering emphasizes systematic development of software systems. Complete development of life cycle phases, including requirements, design, implementation, testing and maintenance, as well as traceability between these phases, is encouraged. Design tools are developed to enforce a uniform structure for specifying the system design, that can be traced up to requirements and down to implementations. Design tools are usually supported by standard methodologies for designing a system, by means of very high level design languages, menus, tables, and graphic notations. Some software engineering tools specify a system in terms of objects, and their relationships and attributes. For each functional component, interface conditions in terms of data and control flow and relationships with other components are given. This information is used for verification and consistency checking and tracing among components of the design. Generally a specific design and control structure is enforced by the tool. For example HOS (Higher Order Software) applies a hierarchical structure [7] and a state-based structure is suggested by Matsumoto [12]. HOS transfers design specifications represented by the functional language AXES to programs in high level languages. In HOS each system is represented by mathematical functions, each function having a specified domain of inputs and range of outputs. A control map is used for interface checking among levels of the functional specifications. Static simulation is used for verification of specifications, and a dynamic simulator provides means for simulating execution of HOS programs. HOS facilitates two levels of transformation, from requirements to design and from design to implementation.

Program Transformation Approach

The program transformation method provides for stepwise refinement and transformation of functional or logical specifications of a system to the implementation. The methods used for the refinement of specifications include rule deduction, theorem proving, and pattern matching. Refinement methods may result in huge amounts of intermediate results. Source-to-source transformation rules are used to simplify and optimize the refinement process. Abstract specifications provide very high level programs at the root of a refinement tree, and applying refinement and source-to-source transformation rules, customized application programs may be provided in a high level language as the leaves of the tree. This method is also sometimes called the stepwise refinement method. Program transformation methods share refinement and transformation methods with different areas of computer science such as artificial intelligence, knowledge-based programming, rapid prototyping, and optimization techniques for compiler construction.

Goldberg [6] has summarized techniques that are applied in program transformation approaches as follows. Stepwise refinement rules mainly include folding and unfolding VHL specifications with the lower level specifications, possibly adding conditions for clarifying VHL concepts in terms of implementations in a high level language. Source-to-source transformations applied for simplification of refinement process including loop optimization, finite differencing, assertion maintenance, algebric or logical simplification, and storage efficiency methods.

The stepwise refinement method is used in the CHI system, [18]. In the CHI system, the language V is used for specification of the design of the system using logical, very high level structure. Logical expressions in the V language, using a pool of generic and instantiated objects, are refined to the lower level constructs of the V language, and finally to LISP. Logic assertion compiler and Rule compiler are used for source-to-source transitions and refinement of specification to the lower level constructs. A data structure synthesizer is used to provide a LISP implementation from generic data objects.

Component Composition Approach

Component composition techniques provide for combination and customization of components from a library of generic components. A system is designed by invoking and interfacing library components and reusing predesigned components. Component composition techniques represent reusable design in its precise and true sense. Due to the fact that a library should be searched for the right component, this method also is referred to as programming by inspection [16]. The adaptable components may be objects representing primitives of the language (e.g., data structure operations, control facilities), and "modules", "plans" or "packages" representing more complex components (i.e., frequently-applied generic modules). For each component some information is provided, such as name, description of functionality, parameters, interface conditions, and rules or axioms for application. A vocabulary set is required for communication between the user and the system for recognition of the library components. Selected components are customized and instantiated by evaluation of their axioms, interface conditions, and generic parameters. Usually a system is designed by decomposition in a top-down fashion to the basic functional components. In order to design a software system the component composition method is used in a manner similar to the bottom-up programming method. Low-level components are customized and combined to provide more complex components from which the last one is the software system. In general the major requirements for implementing this approach include generic design of components, a library, and customization and combination methods.

Numerous studies about human factors in algorithm design and computer programming have suggested that the component composition methods are very close to the human approach [1, 19]. An example of the component composition approach is presented by Goguen [5] in the Library Interface Language (LIL). The language uses very high level generic packages, applying equational logic expressions. Generic packages satisfy "Theories" for their input parameters. Theories provide interface conditions and/or properties of the parameters of the other entities. "Views" show how a given entity (i.e. a package) satisfies a Theory. Finally, the instantiation phase binds the formal parameters to the actual programming language (Ada) data structure. A LIL program is developed by combining, modifying, and importing, using packages or some of their parameters.

APPLICATION-ORIENTED METHODS

Application oriented methods apply reusable designs for producing software systems within a specific application or domain. Applying the domain-specific analysis and software design conventions provides for generation of more efficient software for the domain. Design elements developed in some of these approaches are adaptable in the sense that they represent or apply some classes of objects of the domain.

Knowledge-Based Approach

Knowledge-based methods use domain rules and knowledge, in conjunction with general methods for interpreting the input specifications of a system, and provide some formal or executable form of specifications. Domain analysis may be represented in terms of the software components [11], methods of generating them, theories, rules and experimental facts, domain-dependent refinement rules of specifications, technical names and concepts, and the taxonomy of the domain. This analysis may be used to provide a library of generic components for the domain, for transforming and refining specifications, or for providing methods for deriving more efficient implementations. Though this approach also requires some syntax for input description, requirements are frequently achieved by interactive guidance by the user, using a domain-dependent vocabulary. An important factor about knowledge-based design methods is the role of heuristics in applying domain knowledge and in designing and developing systems. This results in a wide variety of approaches for introducing and applying adaptable designs. An example is an automatic software development system for oil drilling purposes, developed by Schlumberger-Doll Research [3]. The system originally was a problem solver to develop software for solving oil well logging problems. Problem specification is given by a computationally-naive user applying concepts and terms of the domain. Applying stepwise refinement methods and user-defined informal specifications, the system produces a formal design and finally software. Domain knowledge is used for maintaining classification of problems and solutions, recognizing the class of input specification, and providing refinement rules to obtain formal design specifications and implementations.

Application Language Approach

Programming languages use a set of vocabulary and parsing rules to interpret the design of a software system. Tools like lexical analyzers, parsers, and interpreters are based on programming language rules (e.g., BNF), and are used for transforming high level problem representations to machine level code. Software systems developed for specific application domains usually have a set of common concepts including functions, objects, and even problem analysis. These common concepts are used in the syntax of application-oriented languages to allow specifications at a level higher than ordinary programming languages. Similar techniques to the conventional language techniques are used for translation of the programs in application-oriented languages into lower level programs in a programming language. An example of such languages is the simulation language SLAM [15]. SLAM accepts simulation programs and translates them to programs in FORTRAN, and like most other simulation languages has predefined features such as time management, arrival distributions, limited-resource management, and performance data collection. Other examples are graphic languages (packages) that allow higher level descriptions of geometric objects.

Object-Oriented Programming

Different programmers approach software design problems differently. The functional decomposition method emphasizes actions, while data interaction is used as the primary focus for designing a data-centered system. Considering both approaches simultaneously, object-oriented programming views a system or a domain as a collection of objects and their interactions along with their primary functions (methods). This approach allows programming in problem domain concepts rather than machine-oriented programming in terms of variables, memory addresses, operators and operands. Most software design methods somehow deal with objects, their related functions and attributes [9]. Simulation languages come very close to implementing objects and their functions in the manner of object-oriented programming (actually the simulation language SIMULA is considered to be one of the predecessors of the object-oriented languages). The most common definition of an object is an encapsulated data type which can only be accessed through its defined functions or methods [4]. The internal structure of an object is hidden from its users and its functions provide a shell for it. Usually a "message" is used to communicate with an object and to request execution of any of its functions. Most Algol 60 descendant languages that allow definition of data types have the capability to define objects. Encapsulation, concurrent message execution, generic objects, inheritance of objects and methods, libraries of objects, and graphic user-friendly depiction of objects are among the built-in features in the recent object-oriented languages.

Though we have classified object-oriented programming as an application-oriented approach (due to its highly domain dependent application), conceptually it is a general method for designing software systems for any domain. The SMALLTALK language and environment is an integrated system designed on the basis of the object-oriented approach [10]. Everything in SMALLTALK is an object, from numerical types like integers up to entities of the operating system like windows. It allows concurrent message execution for objects of a class, and uses automatic garbage collection for deallocation of resources that may be dynamically bound by messages and are not referenced any longer.

ASSESSMENT OF VERY HIGH LEVEL DESIGN APPROACHES

Software design methods are evaluated from different perspectives. Efficiency, reliability, complexity, degree of automation, and reusability are among the factors that are used here to assess VHL design technique. Emphasis placed by different VHL design methods on each of the above factors varies greatly. Program transformation, in general, requires the user to be able to apply a logical-based or functional-based language. The refinement and transformation process of logical or functional specification is by nature very inefficient [8]. Rule-based refinements require substantial time and storage, and develop huge intermediate results. Refinement deadlock (an intermediate result for which there is no refinement) is another drawback for the program transformation approach. In order to provide a more user friendly environment for obtaining specifications from the user, interface languages are used and translated to the logical/functional design language. This results in a less efficient procedure (compared with other methods) for implementation of the system. In spite of implementation inefficiency of logical or functional-based specifications, the program transformation approach automatically develops full verified implementations and is best suited for verification of designs and for rapid prototyping.

The software engineering approach is based on independent generation and verification of life cycle phases. Specifications at the requirements level can be traced to the design and implementation levels. Design tools are used to standardize design and control structure, and provide reusability of design methodology and structure. Most design tools emphasize interface checking and verification of design specification but do not provide implementation.

Systematic software generation through specification of systems in life cycle phases has been considered in other research than the software engineering approach, per se. Program transformation techniques tend to apply life cycle concepts in their methodologies. Interface languages in these systems play the role of requirement languages and provide consistency checking. On the other hand HOS, one of the very few software engineering tools that claim automatic software generation, implements a functional design language and includes some of the characteristics of the program transformation method. Similar to the component composition method, HOS applies a library of modules for generation of software.

Component composition methods provide efficient means for developing implementations. Considering the degree of reusability and application of predesigned features, the component composition method is preferred to the other general design techniques, especially if combined with knowledge of an application domain. Another advantage of the component composition method is that the internal representation of reusable components can be hidden from the user of these fragments. For example a logic-based language may be used for internal implementation of library components, while the user may use some simple syntax similar to natural language for implementation and instantiation of these components. As mentioned above this is not the case for the program transformation approach. Generic components can be compiled into machine language and saved in the library. These stand-alone standard library components are also referred to as "software ICs" [4]. Like hardware ICs, software ICs can be independently tested, documented, and used for different applications. Hardware ICs, as reusable and encapsulated functional units, have resulted in a revolution for hardware productivity. Though reusable library components may not result in the same revolutionary progress, their application is a milestone in the evolution of the software industry.

Application-oriented approaches in general provide more efficient software for the domain. The interface language applied for specification of the system is closer to the natural languages and applies concepts of the domain. Consequently it is more convenient for users who are familiar with the application domain. The degree of automation, efficiency, and degree of reusability of knowledge-based methods depend on the method used (component composition or program transformation) and the heuristics applied for representing the knowledge of the domain. Some of these systems concentrate on reusability of domain components and improving the productivity of the software generation process [11]. Others emphasize automation and provide rule-based deduction for automatic software generation [3].

Domain language-based design methods allow high level specifications in terms of domain concepts and have resulted in much more efficient implementations. The disadvantages of these languages is their closed view of the application domain. The sets of domain concepts and interpretations are fixed, and language interpreters and parsers have a fix understanding of the domain, which is not extendable. Object-oriented methods, like domain specific languages, allow programming in terms of domain concepts, though they are not as efficient as domain languages. Pure object-oriented programming encapsulates objects, consequently any higher level function needs to be a combination of methods of objects. The resulting code usually is not very efficient and needs optimization.

CONCLUSION

In view of the above comparative analysis, we have become convinced that the greatest practical leverage for reuse can come by a combination of the component composition and application oriented approaches. Component composition methods in general are capable of supporting development of new and complex components from the existing library components more efficiently than other general design methods and can grasp the essence of object oriented programming (that is, designing software in terms of domain concepts), and can enhance the approach and improve its efficiency.

The idea of creation of a single very high level design tool that develops efficient programs for every application domain does not seem to be practical. Representation of programming knowledge in general is not sufficient or efficient for all application domains. Combination of knowledge of application domain and component composition approach develops an open environment for higher level and domain related design of software systems and is thus a step closer to automatic programming.

REFERENCES

1. Adelson, B. and E. Soloway. 1985. "The Role of Domain Experience in Software Design." IEEE Trans. on Software Engineering, Vol. SE-11, no. 11 (Nov.): 1351-1360.
2. Barr, A. and E.A. Feigenbaum. 1982. The Handbook of Artificial Intelligence, Vol. 2. William Kaufman Inc.
3. Barstow, D.R. 1985. "Domain-specific Automatic Programming." IEEE Trans. on Software Engineering, Vol. SE-11, no. 11 (Nov.): 1321- 1336.
4. Cox, B.J. 1986. Object Oriented Programming An Evolutionary Approach. Addison Wesley.
5. Goguen, J. and M. Moriconi. 1987. "Formalization in Programming Environment." Computer, Vol. 20, no. 11 (Nov.): 55-64.
6. Goldberg, A.T. 1986. "Knowledge-Based Programming: A Survey of Program Design and Construction Techniques." IEEE Trans. on Software Engineering, Vol. SE-12, no. 7 (Jul.): 752-768.
7. Hamilton, M. and S. Zeldin. 1979. "The Relationship Between Design and Verification." The Journal of Systems and Software, Vol. 1, no. 1, 29-56.
8. Hoare, C.A.R. 1987. "An Overview of Some Formal Methods for Program Design." Computer, Vol. 20, no. 9 (Sep.): 85-91.
9. Hooper, J.W. 1985. "BPL: A Set-Based Language for Distributed System Prototyping." International Journal of Computer and Information Sciences, Vol. 14, no. 2, 83-103.
10. Key, A. and A. Goldberg. 1977. "Personal Dynamics Media." Computer, Vol. 10, no. 4, (Apr.): 31-41.
11. Lanergan, R.G. and C.A. Grasso. 1984. "Software Engineering with Reusable Design and Code." IEEE Trans. on Software Engineering, Vol. SE-10, no. 5 (Sep.): 498-501.
12. Matsumoto, Y. 1984. "Some Experience in Promoting Reusable Software: Presenting in Higher Abstract Levels." IEEE Trans. on Software Engineering, Vol. SE-10, no. 5 (Sep.): 502-512.
13. Pagan, G.F. 1981. Formal Specification of Programming Languages: A Panoramic Primer. Prentice-Hall.
14. Parnas, D.L. 1985. "Software Aspects of Strategic defense Systems." American Scientist, Vol. 73, no. 5 (Sep.): 432-440.
15. Pritsker, A.A.B., and C.D. Pegden. 1979. Introduction to Simulation and SLAM. Halsted Press, a Division of John Wiley & Sons, Inc..
16. Rich, C. 1984. "A Formal Representation for Plans in the Programmer's Apprentice." M.L. Brodie, J. Mylopoulos, and J.W. Schmidt (eds) On Conceptual Modeling, Chapter9. Springer-Verlag.
17. Simon, H.L. 1986. "Whether Software Engineering Need to Be Artificially Intelligent." IEEE Trans. on Software Engineering, Vol. SE-12, no. 7 (Jul.): 726-732.
18. Smith, D.R., G.B. Kotik, and S.J. Westfold. 1985. "Research on Knowledge-Based Software Environments at Kestrel Institute." IEEE Trans. on Software Engineering, Vol. SE-11, no. 11 (Nov.): 1278-1295.
19. Soloway, E. and K. Ehrilich. 1984. "Empirical Studies of Programming Knowledge." IEEE Trans. on Software Engineering, Vol. SE-10, no. 5 (Sep.): 595-609.
20. Tichy, W.R. 1987."What Can Software Engineers Learn from Artificial Intelligence?" Computer, Vol. 20, no. 11(Nov.):43-54.

N89 - 15569

Artificial Intelligence Approaches to Software Engineering

James D. Johannes, PhD.
James R. Mac Donald

The University of Alabama In Huntsville
Computer Science Department
Huntsville Alabama, 35899

ABSTRACT

This paper examines the artificial intelligence approaches to software engineering. The software development life cycle is a sequence of not so well-defined phases. Improved techniques for developing systems have been formulated over the past 15 years, but pressure continues to attempt to reduce current costs. Software development technology seems to be standing still. The primary objective of the knowledge-based approach to software development presented in this paper is to avoid problem areas that lead to schedule slippages, cost overruns, or software products that fall short of their desired goals.

Identifying and resolving software problems early, often in the phase in which they first occur, has been shown to contribute significantly to reducing risks in software development. Software development is not a mechanical process but a basic human activity. It requires clear thinking, work, and rework to be successful. The artificial intelligence approaches to software engineering presented support the software development life cycle through the use of software development techniques and methodologies in terms of changing current practices and methods. These should be replaced by better techniques that improve the process of software development and the quality of the resulting products. The software development process can be structured into well-defined steps, of which the interfaces are standardized, supported and checked by automated procedures that provide error detection, production of the documentation and ultimately support the actual design of complex programs.

INTRODUCTION

Artificial Intelligence (AI) approaches to software engineering development assist in establishing a knowledge about techniques and methodologies that improve the process of software development and the quality of the resulting products. Given the task of developing a software system, what knowledge is required? To start building of a system of thousand or maybe a million of delivered lines of source code is a daunting prospect. No one should begin without a clear understanding about how the development is to be undertaken. Establishing a software development methodology when undertaking software development, on no matter what scale, is required. Every software organization already has some methodology for building software systems. However, while some software is developed using modern software engineering techniques, most of it is still built in an ad hoc way.

The questions that an AI software engineering system can answer are, what software engineering techniques are there, which are appropriate to our problem at this stage of development, and how can we monitor the quality of the products under development? The knowledge based software engineering paradigm can be summarized as "machine-in-the-loop", where all software project activities are machine mediated and supported. The approach is to assist the programmers rather than replace them. The system acts as an active participant in the software system development process. The system must keep track of details and assist with the routine aspects of the software development life cycle thus allowing the software engineer to concentrate on the more difficult parts.

SOFTWARE ENGINEERING EXPERT SYSTEM

The knowledge based Software Engineering Expert System (SEES) environment can be loosely defined as a computer-based collection of tools, programs, algorithms, etc, which aids in the development of software and/or hardware systems during some phase of the development process. It is a collection of tools, each supporting some part of the software development process, along with tools coordinating and managing the software engineering process [6,9,12,14]. All system development life cycle activities must be machine mediated and supported by the knowledge-based environment as directed by the manager of the project. These activities will be recorded to provide the "knowledge base of software design / programming methods" of the system evolution. These will be used by the SEES to determine how the parts interact, what assumptions they make about each other, what the rationale behind each evolutionary step was, how the project satisfies its requirements, and how to explain all these to the system developers and management of the projects involved. Desirable characteristics for a SEES are:

- supports software using multiple programming languages
- support hardware development for a mixed target-machine complexes
- preserve integration with existing programs and data
- assist all project members (software engineers, managers, technical writers, secretaries, etc.)
- integrated and extensible system knowledge base
- supports component reusability
- user friendly
- supports entire project life cycle with special emphasis on prototyping
- Accommodates multiple projects

This knowledge base SEES is dynamically acquired as a by-product of the development and actual management of each project. It includes not only the individual manipulation steps which ultimately lead to an implementation, but also the rationale behind those development steps. This will make it possible to shift more and more tasks from the software engineer to the machine. To make the process possible, it is necessary to formalize all life cycle activities. In order for the knowledge base software engineering environment to begin to participate in the activities described in the life cycle of the development process (and not just merely record them) the activities must be at least partially formalized.

Formalization is the most fundamental basis for automated support. It creates the opportunity for the environment to undertake responsibility for the performance of the activity, analysis of its effects, and eventually deciding which activities are appropriate. Not only will individual activities become increasingly formalized, but so, too, will coordinated sets of them which accomplish larger development steps. In fact, the development process itself will be increasingly formalized as coordinated activities among multiple projects.

Software Engineering Knowledge Base

A formal software engineering model of definitions and rules that permit a human being to reason about the objects in the this domain, and their interrelations are most desirable, and perhaps necessary, precursor to any techniques for mechanical reasoning and problem solving in the software engineering domain. The knowledge base must contain the knowledge and understanding of the software development process subject matter and incorporate the logical aspect of human intelligence. It must be able to generate problem solutions from situations never before encountered and not anticipated by the software engineering system designers. It must be able to infer the true state of the system from incomplete and/or inaccurate measurements. The knowledge concerning each domain must, at least conceptually, be available in the knowledge base that is used by the various tool reasoning about the current state of the SEES environment.

The type of knowledge required can be divided into two parts: software engineering knowledge and application specific knowledge. The first part is conventional objects of computer science such as control constructs, arrays, sorts, structured programming techniques and their associated algorithms and implementations relationships. This knowledge is the type expressed in computer science text books such as Data Structures and Algorithms [1,15,16,18,23]. The second type is knowledge required about the world in which the target software application is to operate. The system is driven by a database of inexact and judgmental knowledge. Data (knowledge) about the problem domain may be of various forms. Some data may be applicable to the knowledge base; these are generally called (inference) rules since their function is to deduce (new) facts about the domain from the existing data. Other data may take the form of heuristics for deciding when rules or project data can be usefully applied.

The knowledge must be represented in a fashion appropriate for external use and must also be represented internally in such a way that it can be accessed, updated, and efficiently maintained. Several external representations are often desired. For example, the form in which software engineering expert presents knowledge to the knowledge base may differ drastically from the form in which the system represent this information to someone who is not a software engineering expert, a manager, or novice. For the nonexpert, the knowledge would be explained in lay terms, some aspect of the knowledge about certain objects or situations.

In conventional data processing the programmer determines all the relationships among the system modules. AI SEES environment techniques allow the environment itself to determine relationships among the software system symbols that were not made explicit by the programmer. This occurs because the environment has rules for manipulating relationships among symbols whose meanings have be represented within the program by the programmer. This manipulation of relationships among symbols is concerned with preserving not just the data provided but also the knowledge embodied in the relationships among the software elements.

Knowledge Acquisition

Knowledge acquisition is a bottleneck in the construction of SEES environment [13,14]. The SEES knowledge engineer's function is to be a go-between and assist the expert software engineer in building a system that will demonstrate the a level of expertise about the software development process. One of the most difficult aspects of the knowledge acquisition task is helping the software development expert to structure the domain knowledge and to identify and formalize the domain concepts. Potential sources of knowledge include human experts, reports, data bases, and the experience of the software engineers. The knowledge of the software engineering process is subjective, ill-codified, and partly judgmental. The process of extracting knowledge from an software engineer expert during the development process and transferring it to a computer program (expert system) is an important and difficult problem.

Software engineering knowledge acquisition involves problem definition, implementation, and refinement, as well as representing facts and relations acquired from the software development process. The software engineering expert must interact with the SEES environment to build the expertise of the expert system. The main advantages of building an expert system knowledge base are transparency and flexibility. A software engineering expert system knowledge base is developed in two main phases. The first phase is to identify and conceptualize the problem. Identification includes selecting and acquiring a software engineer expert, knowledge source, resources, and clearly defining the software development problem. Conceptualization includes uncovering the key concepts and relations that are needed to characterize the problem. What is the knowledge that software engineers know, and how can it be effectively represented in an SEES? When is the divide and conquer strategy appropriate? For the specific application is the mergesort, the quicksort or selection sort the proper implementation ? The expertise to be elucidated is a collection of specialized facts, procedures, and judgmental rules about a narrow domain area rather than general knowledge about the domain or common sense knowledge about the software development process. The following questions must be answered before proceeding with the next phase:

- What types of data are available?
- What is important in the data interrelations?
- What is given and what is inferred?
- What does a solution look like and what concepts are used in it?
- What aspects of human expertise are essential in solving software development problems?
- What is the nature and extent of "relevant knowledge" that underlies the human solutions?
- Are there identifiable partial hypotheses that are commonly used?
- How do objects in the domain relate?

- Can a diagram of the hierarchy, casual relation, set inclusion, part whole relations, etc., be built?
- What processes are involved in the problem solution?
- What are the constraints on these processes?
- What is the information flow during the software development process?
- Can the knowledge needed to solve a problem be identified and separated from the knowledge used to justify a solution?
- Are the data sparse and insufficient, or plentiful and redundant?
- Is there uncertainty attached to the information?
- Does the logical interpretation of data depend on their order of occurrence over time?
- What is the cost of data acquisition?
- How are data acquired or elicited? What classes of questions need to be asked to obtain data?
- Are the data reliable, accurate, precise; or are they unreliable, inaccurate, or imprecise?

The purpose of knowledge acquisition is to identify and obtain the knowledge needed from a particular software application to be embodied in the SEES environment which is to solve some problem in that application domain. As such, knowledge acquisition is related to the requirements definition phase of a software project. In a traditional software project it is possible to define the proposed system's requirements fully before beginning the design of the software architecture. However in an SEES project this knowledge is not readily available in the same sense as it is in a traditional software project. AI techniques are being employed largely because these techniques lend themselves well to extensive iterative acquisition and refinement of the knowledge from the software engineering and the application domain [3,4,13,14].

The problem that remains, then, is how to minimize the time needed for software system development, and how to make the software development process as effective as possible? Making the process effective involves ways to maximize the amount of knowledge acquired, to maximize the accuracy (in terms of applicability to the project) of the knowledge, and to minimize the effort needed to set up and maintain the software development process. The minimization of software development knowledge acquisition time can be encouraged by providing an environment in which changes are easy, code re-use is easy, turn-around time for changes is quick, and the system is guided down few (if any) dead-end paths during its development. To help focus on the needed knowledge and to maximize accuracy of the knowledge obtained, an early emphasis should be placed on both the overall system's eventual performance and on the knowledge needed to evaluate that performance.

Software Life Cycle Support

A SEES environment must support the software development life cycle. Which system development life cycle should the expert system support? It is very popular to view the software development procedure using the term life cycle [5,8,19,25,26]. The phase seems to be almost a fad or buzz word. There are many representations of the life cycle. Each industry has its own (or several) representations, and each of these tends to be modified somewhat for specific projects. In many cases, the concept of 'life cycle' was used in the sense of 'a suggested ordering of activities in software development, assuming ideal conditions'. This could be the art of programming by trial and error ('hacking'); a method that is unacceptable in professional software engineering. It also covers MIL-STD-490, MIL-STD-1521A, and IEEE Standard for Software Quality Assurance Plans (Std. 730-1981) [17, 18]. Life cycle models are used to emphasize different aspects, e.g. process of development, roles of people involved, etc.. In each case, a life cycle model describes a sequence of steps which may be activities (for instance design, coding, testing, etc.), or is used to clarify the roles for management in the software production process.

A problem exists in representing the coexistence of important aspects of software development. For example, the technical production phase, the management production control, and a step into the application area such as prototype experiments can not be represented. Another problem which some models try to solve is that of directed backtracking. With each phase a set of decisions is associated, of which only one is taken at a given point in time. The decisions, taken in the several phases, are not independent of each other. For both the ongoing development and the maintenance phase it is important to know which decisions belong to a specific stage and where they were taken. As yet, no one has developed a knowledge base project environment that supports a multitude of software life cycles. Most of the systems reviewed in the literature [11,14,24] have their own unique software development life cycle that would have to be integrated into the developers, particular development methodology.

Throughout the reports written on knowledge based software engineering [11,12,14,20,22], there tends to be a severe trivialization of the problem associated with the actual task of translating the specification into code. A significant and disturbing issue brought out in the Kestrel report is that it will take 3 to 6 months of practice before a competent software engineering professional can work with the Knowledge Based Software Assistant (KBSA) system [12]. It appears, based on this statement, that Kestrel Set Theoretic approach leaves much to be desired. Kestral's approach in the KBSA development over the next 3-5 years is minimal and show little thought about the tasks involved in the creation of a reliable knowledge-based system used to develop reliable software systems of the future. The Kestral KBSA system should be able to take advantage of a ten year development cycle. The progress of the hardware revolution will continue (cheaper memory, faster / smaller / interconnected machine's) allowing the KBSA to take advantage of these advances.

Attacking the problem from its high-level esoteric aspects will generate a well thought out specification. The automatic transformation of a high-level prototype into something that can be used as a real system is difficult to imagine. Unfortunately, rapid prototyping rarely discovers those hidden data structures and relationships that are necessary to make a real system operate. Even in sophisticated implementations designed to deal with each aspect of the system's operation, the end result will still rely on the software engineer to design an algorithm that does the job effectively.

People are the highest cost driver attributes in the software development life cycle [7,18]. Shortage of software engineers personnel is between 50,000 and 100,000 people. The suppliers (primarily university computer science departments) do not have sufficient resources to meet the future demand. The demand for a knowledge based SEES environment to aid people in performing their task and coordinating their activities with other members of the team through the knowledge in the system is apparent.

The current life cycle paradigm arose in an era where computers were more expensive than people. There is a need to create a software life cycle paradigm based on automation of the steps within the life cycle. In life cycle models described in the literature, the production phases design, implementation, and test are considered. Phases for requirements analysis and maintenance are often missing. The management of software development is given little or no consideration is almost all life cycle models, with the exception of the first phase of the project planning. The life cycle models look like mappings of three aspects namely management, technical production, and system application or preparation of application. These may be visualize as three simultaneous line of activities, onto one sequence of activities from project conception to system use and maintenance.

Modern programming methodology addresses the difficulties of implementing the chosen application system, not of determining the right system to implement. This has resulted in a batch-oriented development cycle concept predicated on fixing the requirements prior to beginning implementation. This approach assumes that the problem can be correctly determined in detail before a solution is ever seen by the customer. While this approach has been very successful in working around the problems associated with large system implementation efforts which confounded programming teams in the sixties, it avoids the reality dealing with legitimate situations wherein the character of a good solution is itself ill-defined in advance.

The prevailing standard development practices in industry use redundant descriptions to ensure that description mismatches are detected and to guarantee that the implementation corresponds exactly to the original specifications. This serves to freeze the implementation and make it hard to change by accident. It also serves to make the implementation hard to change on purpose if the original specifications are found to be in error.

Phased SEES Development

A possible direction to take is the incremental improvement in each portion of the existing software life cycle for software development. This approach would be a conservative, evolutionary approach described in the "Software Technology in 1990's: Using a New Paradigm" [2]. Because this approach is based on existing software life cycles, the evolutionary approach is limited by any weakness of that life cycle. Existing life cycles are not considered to be good candidate for an SEES environment because of two fundamental flaws that aggravate the maintenance problem. There appears to be no technology formalism associated with managing the knowledge intensive activities that constitute the software development process. The life cycle models reviewed in literature are informal, labor intensive, and largely lack formal documentation. Information about what specific process occurred during each phase of the development and the rational behind each decision is crucial for the maintenance process. During the software development process programming skill is applied to optimize the source code. This optimization over time makes the maintenance problem harder by making the software harder to understand [21]. The increasing dependencies among the components and scattering related design decision information about the development process over time requires machine mediation.

In the SEES approach of supporting the current life cycle models, rather than making a major revision to the activities and products of the life cycle, the existing life cycle elements and their interaction are examined for possible use of knowledge based tools. Carnegie Group and Boeing Computer Services are building a knowledge based software development environment based on this approach [20]. The environment supports the software engineer and project management using artificial intelligence. The system will provide a framework in which conventional software tools can be integrated with tools based on AI. The objectives of these efforts is to increase software engineering productivity. Currently the knowledge base software engineering environments are in their infant stages of development.

Software project management has the responsibility for planning, controlling and coordinating software life cycle activities. Currently, project managers are hampered by the informal and undocumented nature of the activities, and the fragmentary, obsolete, and inconsistent data available. More effective project management requires not only improved management techniques, but also a better software development environment that captures the total project life cycle activities and the rational behind the development process for a project. The knowledge base SEES is an intelligent environment (or collection of environments) which aids personnel in performing their tasks, and coordinates their activities with other members of the team.

TRW's Distributed Computing Design System (DCDS) provides an integrated set of environments for development of real-time distributed software systems [10]. The primary focus of DCDS is to improve system reliability, software productivity, and to minimize schedule and cost risks. Unlike the work done at Kestrel, the DCDS is strongly focused on those aspects of distributed processing involving component interaction, function architecture pairing, data distribution, deadlock avoidance and system recovery. The approach to DCDS is to define the different phases of the software development life cycle in terms of different languages, with each language specifically designed to support that aspect of each life cycle process. Information is passed between these languages through a common database and interface specification. The DCDS design is currently based on five languages and methodologies, specifically designed to attack: System requirements, Software requirements, Distributed design, Module development, and Test support. DCDS has two key aspects that it shares with Knowledge-based systems: the central database that collects all documentation form requirements to code and test cases, and the use of specialized languages designed for specific problems. Knowledge based systems support both a central knowledge base and a very high level but wide spectrum language. If the DCDS languages are taken together, they form the basis of a wide spectrum language.

CONCLUSION

This paper has examined an artificial intelligence approach to software engineering. The software development life cycle has been presented as a sequence of not so well-defined phases and as such presents a major hurdle in SEES development. Improved techniques for developing systems have been formulated over the past 15 years, but shortcuts continue to be exercised in attempts to reduce current year costs. In this sense, software development technology seems to be standing still. The SEES approach will reduce the software development problem areas that lead to schedule slippages, cost overruns, or software products that fall short of their desired goals.

A knowledge based SEES approach to the software development process will someday become a reality. However, many industry practitioners are crating new problems in trying to solve old ones. The selection of a new specific life cycle model for software development has the danger of making the problem just as unsolvable after its introduction as it was before. Will new paradigms for software development give the necessary productivity gain? Will the cost of their implementation cause the total development cost to exceed that of the development via the traditional models? These knowledge base software engineering systems will not be trivial to learn. Training, on the order of weeks months will be required to achieve acceptable efficiency in the production of software systems. The results will be a higher system reliability and maintainability as well as present less risk to the system developer.

A primary difference between artificial intelligence and more traditional ADP approaches is summarized by the slogan "In the Knowledge Lies the Power." The operative word is knowledge, rather than data or processor speed. Knowledge intensive systems attempt to model the imperfectly-understood decision processes of the domain practitioner and, like the human practitioner, make decisions with less than certainty.

BIBLIOGRAPHY

1. Aho, A. V., J. E. Hopcroft, and J. D. Ullman, "Data Structures and Algorithms", Addison-Wesley, 1983.

2. Balzer, R., et al., "Software Technology in the 1990's: Using a New Paradigm", IEEE Computer, November 1983.

3. Barr, Avron and E.A. Feigenbaum, "The Handbook of Artificial Intelligence, Volume 1, William Kaufmann, Inc., Los Altos, Ca., 1981.

4. Barr, Avron and E.A. Feigenbaum, "The Handbook of Artificial Intelligence, Volume 2, William Kaufmann, Inc., Los Altos, Ca., 1982.

5. Boehm, B. W. "Software Life Cycle Factors," TRW Software Series, Jan 1981.

6. Boehm, B. W., et al., "A Software Development Environment for Improving Productivity", IEEE Computer, June 1984.

BIBLIOGRAPHY (Continued)

7. Bruce, P. and S. M. Pederson. " The Software Development Project: Planning and Management", NY: John Wiley and Sons, 1982.

8. Daly, E., "Management of Software Development," IEEE Transactions on Software Engineering, May 1977.

9. Davis, C. G. and C. R. Vick, "The Software Development System," IEEE Transactions on Software Engineers, Jan 1977, vol 3, num 1.

10. --,DCDS A Unified Environment for System Software Development", Summary Description, Volume 1, TRW, Huntsville, AL., January 1987.

11. Goldberg, A. "Knowledge-based Programming: A Survey of Programming Design and Construction Techniques", Kestrel Institute, Palo Alto ,Ca., July 1986.

12. Green, C. et al., "Report on a Knowledge-based Software Assistant", Kestrel Institute, Palo Alto, Ca., June 1983.

13. Hayes-Roth, F., et al., "Building Expert Systems", Addison-Wesley Publishing Company, Inc., 1983.

14. Harandi, M. T., " Applying Knowledge-Based Techniques to Software Development," Perspective in Computing, 6(1), 14-21, 1986.

15. Horowitz, E., Sahni S., "Fundamentals of Data Structures", Computer Software Press, Inc., 1982.

16. Horowitz, E., Sahni S., "Fundamentals of Computer Algorithms", Computer Software Press, Inc., 1984.

17. IEEE Computer Society. IEEE Standard for Software Quality Assurance Plans, NY: IEEE, Inc, 1982.

18. Jensen, R. and C Tonies, "Software Engineering", Englewood Cliffs,NJ: Prentice-Hall, 1979.

19. Kerola, P. and P. Freeman, "A Comparison of Lifecycle Modles," IEEE Computer Society, Fifth International Conference on Software Enginerring , San Diego, 1981, pp 90-99. Siler Spring, MD: IEEE, Inc,1981.

20. --,"Knowlege-based Software Development Envrionment, Carnegie Group Inc., Augest 1985.

21. McClure, C., " Managing Software Development and Maintenance", NY, Van Nostran Reinhold Ltd, 1981.

22. Smith, D.R., et al.,"Research on Knowledge-Based Software Environments at Kestral Institute", IEEE Transactions on Software Engineering, November 1985.

23. Sommerville, I. "Software Engineering", London: Addison- Wesley Publishers Limited, 1982.

24. Swanson, E. B. "The Dimensions of Maintenance," Tutorial; Automated Tools For Software Engineering, NY: IEEE Inc, pp 240-245.

25. Teichroew, D., "Improvements in the System Life Cycle," Tutorial on Software Design Techniques, San Framcisco: IEEE, Inc, 1976 pp 64-70.

26. Zvegintzov, N., "What life? What cycle?" AFIPS Conference Proceedings, 1982 National Computer Con-ference,Houston, 1982, pp 561-568. Arlington, Va: AFIPS Press 1982.

N89-15570

Automatic Programming for Critical Applications

Raj L. Loganantharaj
The Center for Advanced Computer Studies
USL, P. O. Box 44330
Lafayette, LA 70504

(Extended Abstract)

The important phases of a software life cycle includeverification, and maintenance. Usually, the execution performance is an expected requirement in a software development process. Unfortunately, the verification and the maintenance of programs are the time consuming and the frustrating aspects of software engineering. The verification cannot be wavered for the programs used for critical applications such as, military, space, and nuclear plants. As a consequence, synthesis of programs from specifications, an alternative way of developing correct programs, is becoming popular.

The definition, or what is understood by automatic programming, has been changing with our expectations. At present, the goal of automatic programming is the automation of programming process. Specifically, it means the application of artificial intelligence to software engineering in order to define techniques and create environments that help in the creation of high level programs. The automatic programming process may be divided into two phases: the problem acquisition phase and the program synthesis phase. In the problem acquisition phase, an informal specification of the problem is transformed into an unambiguous specification while in the program synthesis phase such a specification is further transformed into a concrete, executable program.

We propose the automation of the design and the programming of software systems for critical applications as a long term goal of computer aided software engineering. To realize the long term goal, we have to have a good understanding of the automation of the programming. We believe that the automation of Logic Programming provides a good understanding to pursue the long term goal. We have selected Logic program as the target language to narrow the semantic gap between the specification and the program.

In our approach, the program specified in the first order logic is transformed into a logic program (a set of Horn clauses) by the repeated application of nested resolution. Our approach is similar to Varghese's approach. The nested resolution could be viewed as the generalization of a set of Varghese's transformation rules.

The desired program is described in first order logic. This specification forms the axioms and is sometimes called the specification set. The axiom set also includes the generic axioms.

Inference rules are applied to a pair of statements: a statement to be transformed which we call a transformee, and a statement used for transformation which we call a transformer. The transformer may be an axiom, a transformation rule or a lemma. The transformee is initially an axiom from the specification set and subsequently it may be a lemma or even a transformation rule. The subwffs that are replaced are always in the transformee. Program derivation starts with an axiom from the specification set. This statement is incrementally transformed using other axioms from the specification set, generic axioms, transformation rules and any lemmas that might have already been derived. The application of the inference rules continues until the desired Horn clauses have been derived or no more Horn clause can be derived.

The realization of automatic programming is challenging. However, the recent development in automatic programming, that is the availability of nonclausal theorem proving technology and the progress in nested resolution make easy to bridge the semantic gap between the specification and the target language. To understand the basic control issues in deductive program synthesis we have selected Logic Programming as a target language (with this selection the semantic gap is somewhat reduced).

N89-15571

Using Automatic Programming for Simulating Reliability Network Models

Fan T. Tseng
Bernard J. Schroer
University of Alabama in Huntsville
Huntsville, Alabama, USA 35899

S. X. Zhang
Northwestern Polytechnical University
Xian, Shaanxi, China

John W. Wolfsberger
NASA Marshall Space Flight Center
Marshall Space Flight Center, Alabama, USA 35812

ABSTRACT

This paper presents the development of an automatic programming system for assisting modelers of reliability networks define problems and then automatically generate the corresponding code in the target simulation language GPSS/PC.

INTRODUCTION

There has always been a desire of software developers to automate more and more of the computer programming process. The goal of these developers has been to have a system that can carry on a natural language dialogue with the user in defining his problem and then to automatically generate the appropriate computer code. The term automatic programming (AP) has been defined as an application of artificial intelligence (AI) in automating some aspects of the computer programming process (Barr and Feigenbaum 1982). This automation is generally accomplished by developing another program, an AP system, that raises the level of specifying computer program instructions. In other words, an AP system is a program that helps programmers write programs.

An AP system should improve the overall environment for defining and writing the program (Brazier and Shannon 1987). As a result of this improved environment, there should be a reduction in the amount of detail that the programmer needs to know. Quite possibly, the user could even do his own programming with the help of an AP system. Also, this improved environment should result in a more natural way for the user to define his problem that closely resembles the user's way of thinking and looking at problems.

RESEARCH GOAL

The goal of the research presented in this paper is to develop an AP system to assist the modeler of reliability networks define problems, and to then automatically generate the program code in the target simulation language GPSS/PC. The AP system is called Automatic Network Programming System (ANPS).

The domain of problems that can be solved by ANPS include prelaunch activities of space vehicles, operation of ground support equipment, space vehicle turn around activities, space transportation systems and operational plans, and hardware system with multiple subsystems. The ANPS system requires that the problem be defined by:

- A network of activities with starting and stopping events.
- Activities with either fixed or continuous times.
- Activity failures and repairs (mean times to failure and repair).
- Operational dependencies between activities.

PREVIOUS RESEARCH

Synder et al. (1967) developed a simulation model of the Saturn V prelaunch activities beginning at T-24 hours and continuing through T-0 hours, or lift-off. This model was used to predict launch vehicle availability (LVA). LVA was defined as the probability of launching the spacecraft within a given launch window. A second objective of the model was to identify locations in the countdown for placing holds and to determine the length of these holds.

The Synder model consisted of over 1100 vehicle subsystems and 400 ground support subsystems. A detailed time line was developed showing the interrelationships of these subsystems. In addition to the time line, the model input included operational data, reliability data, and maintenance data. The model was written in GPSS-II and ran on an IBM 360 computer.

The original Synder model was expanded to include multiple launch windows and the operational sequence when a launch window was missed and the spacecraft had to be recycled to the next launch window (Schroer 1969). The model was used to predict the probability of launching a spacecraft within a given set of back-to-back launch windows. A second objective was to predict the probability of launching in a subsequent window, given a window had been missed and a recycle sequence and a possible hold had to be executed before resuming the countdown.

The expanded model included two countdown sequences. The first sequence was the main countdown sequence identical to the Synder model. The second sequence was the recycle sequence that consisted of a number of backout sequences containing those events that were required to return the countdown to some preceding point. The recycle sequence also consisted of a recycle hold containing those activities that were required to sustain the vehicle status at a particular time in the countdown. The model was written in GPSS-II, contained 2300 blocks, several Fortran help routines and ran on the IBM 360 computer.

A goal of the ANPS system is to be able to model these types of applications more quickly and accurately than previously done using conventional simulation techniques.

ANPS SYSTEM

Figure 1 gives an overview of the ANPS system. The ANPS system is designed using the elements of automatic programming as its foundation. The three AP elements in ANPS are; an interactive user dialogue interface, a library of software modules, and an automatic simulation code generator. In Figure 1, the traditional programming task of flow charting has been replaced by the interactive user dialogue interface and the problem specification. Likewise, the program writing task in Figure 1 has been replaced by the automatic code generator and the library of software macros. The ANPS system is written in Turbo Prolog (Borland 1986) on an IBM PC class of personal computer. The system contains 1218 lines of code and 86 subroutines. The simulation code generated by ANPS is GPSS/PC (Minuteman 1986).

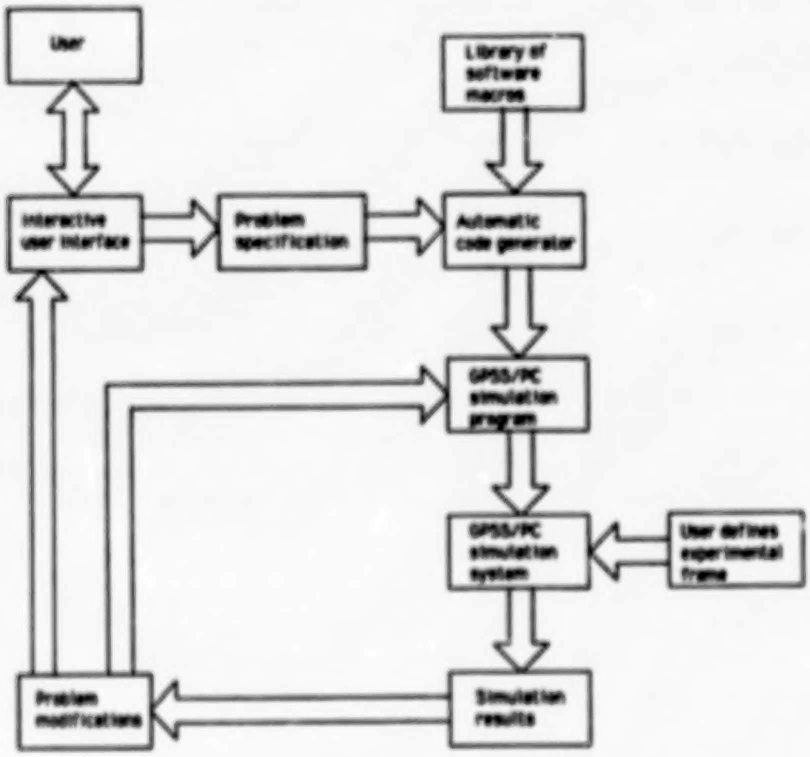

User
Library of software macros
Interactive user interface
Problem specification
Automatic code generator
GPSS/PC simulation program
GPSS/PC simulation system
User defines experimental frame
Problem modifications
Simulation results

Figure 1. ANPS system overview

Interactive User Dialogue Interface

There are three commonly used AP user interfaces. These are a natural language interface, a graphical user interface, and an interactive dialogue interface. Several natural language interface developments are Heidorn (1974) and Ford and Schroer (1987). An example of a graphical interface development is Khoshnevis and Chen (1986). Several interactive dialogue interfaces are Haddock and Davis (1985), Brazier and Shannon (1987), and Murray and Sheppard (1988).

The ANPS system uses an interactive dialogue interface. This interface is probably the most common and easiest to develop interface. Using this interface, the user, or modeler, enters into a dialogue with the ANPS system to define the problem specification, or model.

Library of Software Modules

The robustness of an AP system is dependent on the diversity and completeness of its library of software modules. Furthermore, this library is generally domain specific. When new modules or subroutines are needed, expert simulation programmers are needed to write the simulation code and to assure the proper interface.

Since the ANPS system is domain specific to system reliability networks, the number of needed software modules is minimal. At this point of development, ANPS consists of the following four modules:

- Fixed activity operation function
- Variable activity operation function
- Activity failure function
- Activity interrupt function

These modules were selected based on a detailed evaluation of the two previously discussed models by Synder (1967) and Schroer (1969). Interestingly, several of these previously developed modules were written as Fortran HELP routines using the old GPSS-II.

The fixed activity operation function (VENT_A) simulates the operation of each fixed time activity and its time to failure. If the activity fails during its operation, the transaction is forwarded to the activity failure function (FAIL).

The variable activity operation function (VENT_B) simulates the operation of each variable time activity and its time to failure. This activity is not completed until all other related activities are completed. For example, system power is a variable time function that will be on until all activities requiring power are completed. If the activity fails, the transaction is forwarded to the activity failure function (FAIL).

The activity failure function (FAIL) simulates the failure of an activity as indicated by functions VENT_A and VENT_B. When an activity fails, all the dependent activities enter a hold state. The function then simu-lates the time to repair the activity. If another activity fails during the delay of a dependent activity and the dependent activity is dependent on the first failed activity, the additional time to repair, if any, is added to the delay of the dependent activity. The function assumes that a dependent activity that has been delayed cannot fail during the delay. The activity interrupt function XACT_DELAY contains the logic to add any additional time to an activity on hold if another activity fails during the hold and the held activity is dependent on the failed activity.

Figure 2 is a listing of the GPSS code for the fixed activity function VENT_A. Note that the subroutine makes extensive use of indirect addressing. The system also contains a large number of matrix savevalues for transferring data between the subroutines and the main program. Initially, all the input data from the problem specification are entered into these matrix savevalues.

Automatic Simulation Code Generator

The output from the interactive dialogue interface, or the problem specification, is then used as input to the code generator program which automatically writes the program code in the target simulation language GPSS. The system creates the main program that includes the appropriate calls to the selected library macros.

```
*
*        ACTIVITY TIME SIMULATION GENERATOR
*
VENT_A SEIZE        P2
       ASSIGN       ETIME,MX$WORK_TIME(P3,1)
BACK3  ASSIGN       MTTF,MX$F_TIME(P3,1)
       TEST L       P$MTTF,P$ETIME,NOFAIL
       ADVANCE      P$MTTF
       ASSIGN       ROW,P3
       TRANSFER     SBR,FAIL,RTRN1
       ASSIGN       REST_TIME,V$TIME3
TIME3  FVARIABLE    P$ETIME-P$MTTF
       ASSIGN       ETIME,P$REST_TIME
       TRANSFER     ,BACK3
NOFAIL ADVANCE      P$ETIME
       RELEASE      P2
       TRANSFER     P,RTRN2,1
```

Figure 2. GPSS listing for function VENT_A

SAMPLE PROBLEM

Figure 3 is a time line for a typical network consisting of nine fixed activities and two variable activities. Figure 4 is the time line redrawn in the form of a network diagram. Activity 12 is a dummy activity with the time equal to zero.

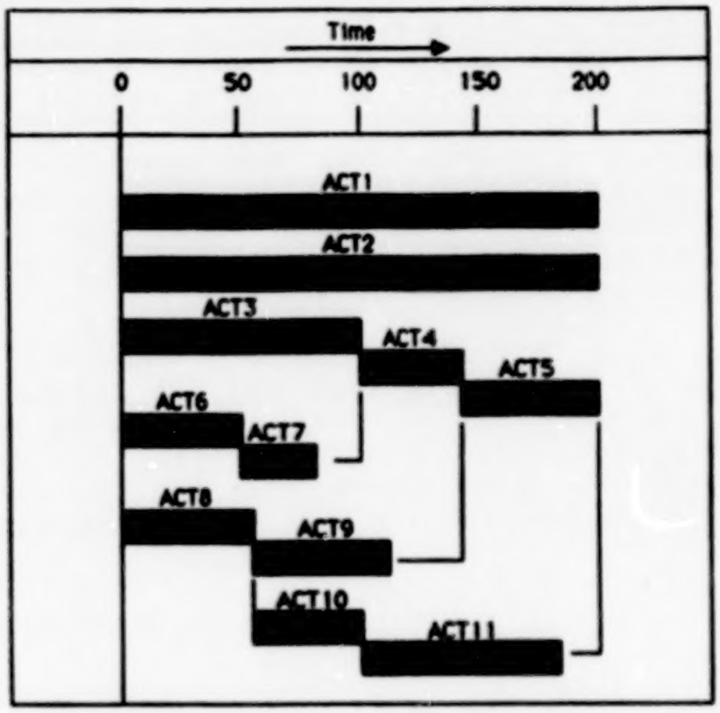


Figure 3. Sample problem time line

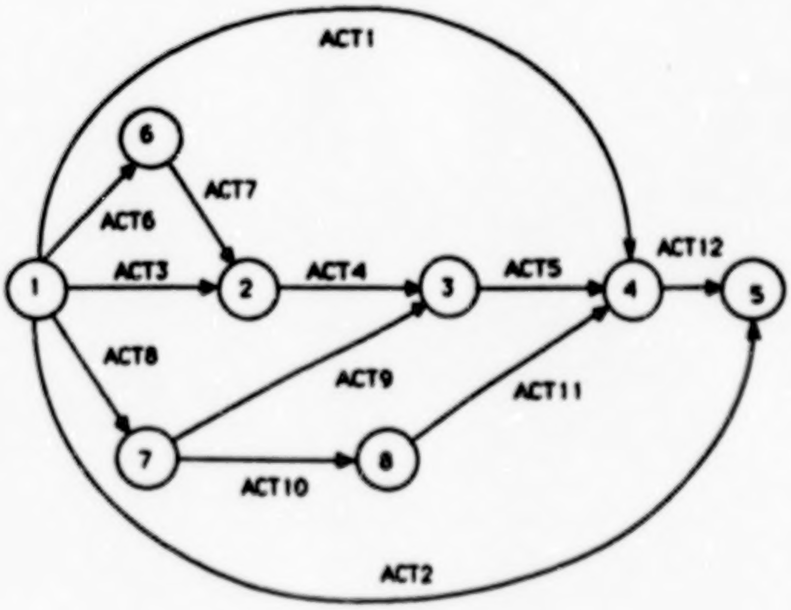


Figure 4. Sample problem reliability network

Table I gives the time parameters for the activities. These parameters include activity duration, time to failure and time to repair. Activities ACT1 and ACT2 have variable times; therefore, the activities will operate during the entire duration of the system. Table II gives the operational dependencies between the activities. For example, a failure of activity ACT1 will cause a stopping of activities ACT3, ACT4, and ACT5. Likewise, a failure of activity ACT9 will also stop activity ACT5.

Table I. Activity Time Parameters

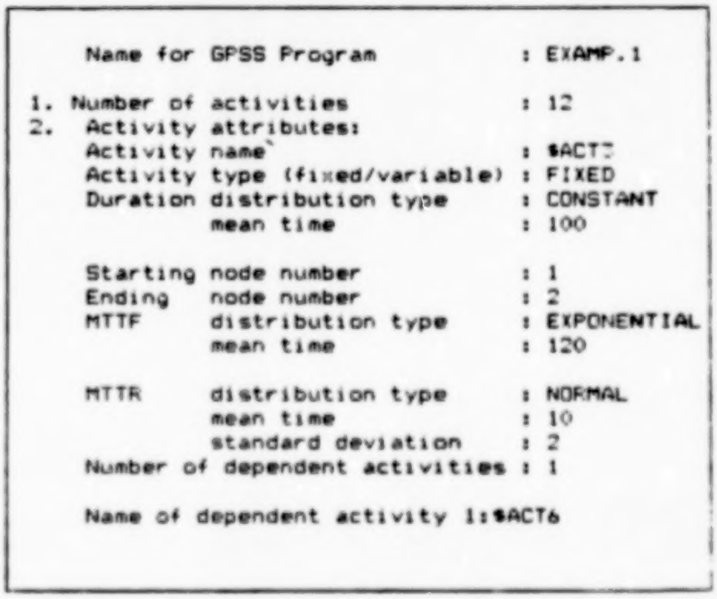
Activity	Duration	Time to Failure	Time to Repair
ACT1	Variable	E(200)	N(20,4)
ACT2	Variable	E(200)	N(20,4)
ACT3	100	E(120)	N(10,2)
ACT4	40	E(60)	N(5,1)
ACT5	60	E(80)	N(5,1)
ACT6	50	E(60)	N(5,1)
ACT7	30	E(100)	N(10,2)
ACT8	60	E(100)	N(10,2)
ACT9	60	E(50)	N(5,1)
ACT10	40	E(60)	N(5,1)
ACT11	80	E(100)	N(10,2)
ACT12	Dummy	0	0

Table II. Operational Dependencies Between Activities

	Dependent Activities											
Activity	ACT1	ACT2	ACT3	ACT4	ACT5	ACT6	ACT7	ACT8	ACT9	ACT10	ACT11	ACT12
ACT1	X		X	X	X							
ACT2		X						X	X			
ACT3			X			X						
ACT4				X								
ACT5					X							
ACT6						X						
ACT7							X					
ACT8								X				
ACT9					X				X			
ACT10										X		
ACT11											X	
ACT12												X

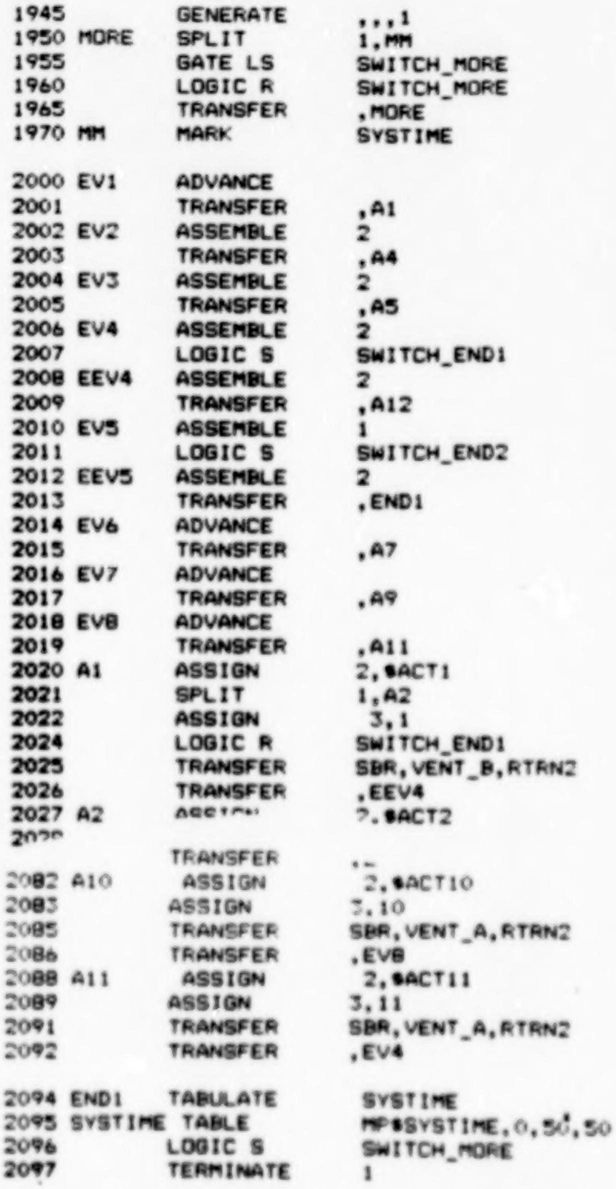
Figure 5 is a partial listing of the interactive user dialogue for defining activity ACT3. ACT3 starts at node 1 and ends at node 3. The activity type is fixed, the duration is 100, the time to failure follows an exponential distribution with a mean of 120, the time to repair the activity follows the normal distribution with a mean of 10 and a standard deviation of 2. Activity ACT6 is dependent on ACT3.

Figure 6 is a partial listing of the GPSS main program. Note that the code consists of a number of SPLIT, TRANSFER and ASSEMBLE blocks that define the network. The TRANSFER blocks route the transactions to the appropriate fixed or variable activity operation subroutines.



```
    Name for GPSS Program                 : EXAMP.1

 1. Number of activities                  : 12
 2.  Activity attributes:
    Activity name`                        : $ACT3
    Activity type (fixed/variable) : FIXED
    Duration distribution type            : CONSTANT
             mean time                    : 100

    Starting node number                  : 1
    Ending   node number                  : 2
    MTTF     distribution type            : EXPONENTIAL
             mean time                    : 120

    MTTR     distribution type            : NORMAL
             mean time                    : 10
             standard deviation           : 2
    Number of dependent activities : 1

    Name of dependent activity 1:$ACT6
```

Figure 5. Partial listing of interactive user dialogue



```
1945          GENERATE      ,,,1
1950 MORE     SPLIT         1,MM
1955          GATE LS       SWITCH_MORE
1960          LOGIC R       SWITCH_MORE
1965          TRANSFER      ,MORE
1970 MM       MARK          SYSTIME

2000 EV1      ADVANCE
2001          TRANSFER      ,A1
2002 EV2      ASSEMBLE      2
2003          TRANSFER      ,A4
2004 EV3      ASSEMBLE      2
2005          TRANSFER      ,A5
2006 EV4      ASSEMBLE      2
2007          LOGIC S       SWITCH_END1
2008 EEV4     ASSEMBLE      2
2009          TRANSFER      ,A12
2010 EV5      ASSEMBLE      1
2011          LOGIC S       SWITCH_END2
2012 EEV5     ASSEMBLE      2
2013          TRANSFER      ,END1
2014 EV6      ADVANCE
2015          TRANSFER      ,A7
2016 EV7      ADVANCE
2017          TRANSFER      ,A9
2018 EV8      ADVANCE
2019          TRANSFER      ,A11
2020 A1       ASSIGN        2,$ACT1
2021          SPLIT         1,A2
2022          ASSIGN         3,1
2024          LOGIC R       SWITCH_END1
2025          TRANSFER      SBR,VENT_B,RTRN2
2026          TRANSFER      ,EEV4
2027 A2       ASS           2.$ACT2
20
             TRANSFER      ,
2082 A10       ASSIGN        2,$ACT10
2083          ASSIGN        3,10
2085          TRANSFER      SBR,VENT_A,RTRN2
2086          TRANSFER      ,EV8
2088 A11       ASSIGN        2,$ACT11
2089          ASSIGN        3,11
2091          TRANSFER      SBR,VENT_A,RTRN2
2092          TRANSFER      ,EV4

2094 END1     TABULATE       SYSTIME
2095 SYSTIME TABLE           MP$SYSTIME,0,50,50
2096          LOGIC S        SWITCH_MORE
2097          TERMINATE      1
```

Figure 6. Partial GPSS listing of main program

CONCLUSIONS

The ANPS system is currently in limited operation on an IBM PC microcomputer. A number of relatively small network problems have been solved using the system. Given the success in modeling these small networks, it appears that the ANPS system can readily model the two large Saturn V prelaunch models by Synder (1967) and Schroer (1969). Based on this initial testing and evaluation, the following comments can be made:

- The interactive user dialogue provides for a formal and structured procedure for acquiring information on the network being modeled.
- The interactive user dialogue expedites the definition of the problem specification and assures a complete and detailed definition of the problem specification.
- The automatic code generator results in structured simulation code that is easy to read, trace and modify.
- The overall clarity of the simulation code is greatly improved.
- The ANPS system is ideal for rapid prototyping and can produce simulation code that is syntax error free.
- The ANPS system reduces the knowledge level required by the modeler of the simulation language.

The ANPS system also has several disadvantages. These disadvantages include:

- The system is domain specific and limited by the robustness of its library of macros.
- The GPSS code generated by ANPS probably is longer, and consequently requires more memory and takes longer to execute, than a nonstructured equivalent program.

A second version of ANPS is currently under development on an Apple Mac II using HyperCard. This version uses an interactive graphical interface rather than the interactive user dialogue. With this version it will be possible to compare the different interface approaches to defining the problem specification, the use of Turbo Prolog versus HyperCard, and the PC and Mac II platforms.

ACKNOWLEDGEMENTS

This research was funded in part by grant NAG8-641 from the NASA Marshall Space Flight Center and contract ADECA-UAH-9001 from the Science, Technology, and Energy Division of the Alabama Department of Economic and Community Affairs.

REFERENCES

Barr, A. and E. A. Feigenbaum, 1982, *The Handbook of Artificial Intelligence*, Vol. 2, W. Kaufman, Inc., CA.

Brazier, M. K. and R. E. Shannon. 1987. "Automatic Programming of AGVS Simulation Models," *1987 Winter Simulation Conference*, Atlanta, GA, (December) pp. 703 - 708.

Ford, D. R. and B. J. Schroer. 1987. "An Expert Manufacturing Simulation System." *Simulation*, Vol. 48, No. 5, (May) pp. 193-200.

GPSS/PC Reference Manual, 1986, Minuteman Software, Stow, MA.

Haddock, J. and R. P. Davis. 1985. "Building a Simulation Generator for Manufacturing Cell Design and Control." *Annual International Industrial Engineering Spring Conference Proceedings*, Los Angeles, CA, (May) pp. 237-244.

Heidorn, G. E. 1974. "English as a Very High Level Language for Simulation Programming." *SIGPLAN Notices*, Vol. 9, No. 4, pp. 91-100.

Khoshnevis, B. and A. P. Chen. 1986. "An Expert Simulation Model Builder." Intelligent Simulation Environment, Society for Computer Simulation, Vol. 17, No. 1, pp. 129-132.

Murray, K. J. and S. V. Sheppard. 1988. "Knowledge-based Simulation Model Specification," *Simulation*, Vol. 50, No. 3, (March) pp. 112-119.

Schroer, B. J. 1969. "Saturn V Prelaunch Systems Simulation Model for a Launch Opportunity Containing Multiple Launch Windows," *Third Conference on Applications of Simulation*, Los Angeles, (December) pp. 503-511.

Synder, J. E., E. R. Bennich and Y. H. Lindsey. 1967. "Implementation of Advanced Simulation Techniques for Predicting the Saturn V Launch Vehicle System Behavior," *Journal of Spacecraft and Rockets*, Vol. 4, No. 8, pp. 998-1002.

Synder, J. E. R. Bennich and Y. H. Lindsey. 1967. "Implementation of Advanced Simulation Techniques for Predicting the Saturn V Launch Vehicle System Behavior," *AIAA 5th Aerospace Sciences Meeting*, Paper 67-205, New York, January 1967.

Turbo Prolog 2.0 Reference Guide. 1986. Borland International, Scotts Valley, CA.

N89-15572

OBJECT ORIENTED STUDIES INTO ARTIFICIAL SPACE DEBRIS

J. M. Adamson,
Consultant, 1 Brockhurst Cottages, Over Wallop, Stockbridge,
Hampshire, England

G. Marshall,
Consultant, 31 Broomhill Way, Allbrook, Eastleigh,
Hampshire, England

ABSTRACT

A prototype simulation is being developed under contract to the Royal Aerospace Establishment (RAE), Farnborough, England, to assist in the discrimination of artificial space objects/debris.

The methodology undertaken has been to link Object Oriented programming,intelligent knowledge based system (IKBS) techniques and advanced computer technology with numeric analysis to provide a graphical, symbolic simulation. The objective is to provide an additional layer of understanding on top of conventional classification methods.

Use is being made of object and rule based knowledge representation, multiple reasoning, truth maintenance and uncertainty. Software tools being used include Knowledge Engineering Environment (KEE) and SymTactics for knowledge representation. Hooks are being developed within the SymTactics framework to incorporate mathematical models describing orbital motion and fragmentation. Penetration and structural analysis can also be incorporated.

SymTactics [15] is an Object Oriented discrete event simulation tool built as a domain specific extension to the KEE environment. The tool provides facilities for building, debugging and monitoring dynamic (military) simulations.

INTRODUCTION

There are currently some 10,000 registered objects orbiting the Earth. It is estimated that another 40,000 golf-sized objects are not tracked along with billions of still smaller pieces. These objects consist of satellites, extinct rocket casings and debris.

Space debris can be effectively categorised under the headings of : particles, fragments and artifacts [10].

Many tiny particles are produced by solid rocket motors used in space. Larger particles can be attributed to the intense thermal cycling of the space environment (for example paint flaking off satellites). Particles are likewise produced by explosions, both accidental and deliberate.

A principle source of fragments and particles is the destruction of spent rocket stages and satellites. Explosions of rocket stages can occur many months or years after launch. Residual propellants may be the cause here. Spacecraft on the other hand tend to be destroyed deliberately. This may result from testing anti-satellite weapons, or spacecraft being commanded to self-destruct for various reasons.

Numerous parts of spacecraft are jettisoned during launch and operation; these come under the fragment category. Fragments of this type include interstage structures, payload shrouds and support structures. Fragments are likewise produced during rocket stage separations.

The term artifacts is applied to derelict items of space hardware such as intact payload support structures, spent upperstages and spacecraft. Spacecraft can become derelict (non-operational) following the malfunction of a launch vehicle/upper stage, insertion into an incorrect orbit or following a system malfunction.

A major concern exists that space debris may cause collisions, particularly in orbits ranging from 500km to 1100km above the Earth. These collisions can produce many small fragments which in turn increase the probability of further collisions.

Space scientists and mission planners are becoming increasingly concerned about the possibility of space objects (namely spacecraft, rockets, spaceplanes and space stations) being damaged by artificial space debris. Similar problems exist, though not to the same extent, when placing telecommunication satellites into geostationary orbit.

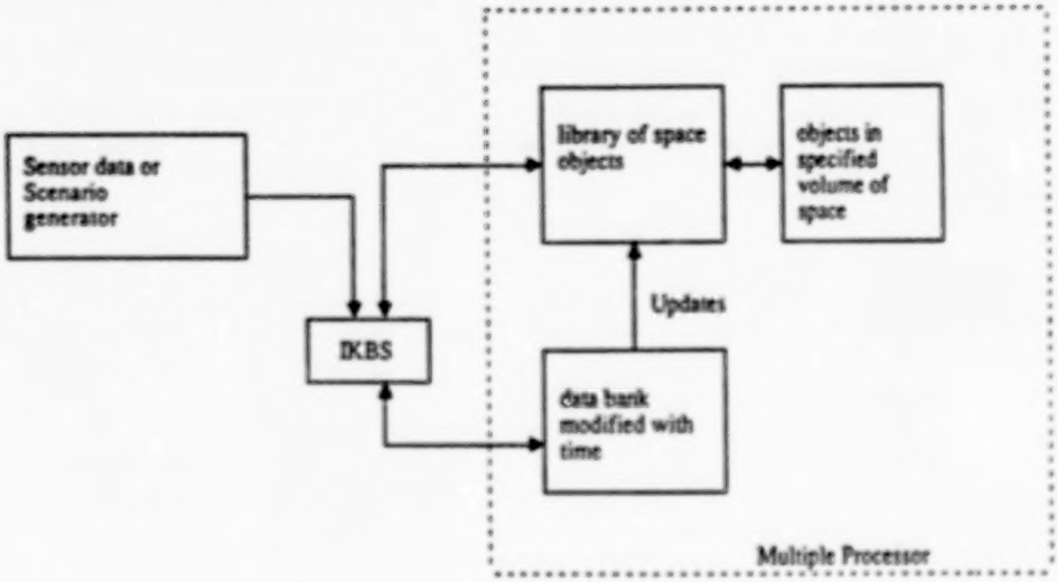
Should an operational space object become non-operational, it is in the interest of space scientists and mission planners to determine the cause of this event as soon as possible. If space debris is suspected, the resultant signature data and detailed simulations could identify the nature and cause of this event.

SYSTEM CONFIGURATION

The proposed system configuration for this simulation (see Fig. 1) makes use of a library of space objects, intelligent sensors and a scenario generator. When a sensor detects an object, it interrogates the library of space objects to determine if that object has been identified/categorized. If not, the sensor reasons as to the possible cause and origin of the unidentified object.

The first phase of this prototype simulation is currently being modelled on a Lisp machine. The computer used was a Symbolics 3640. The simulation is now in the process of being

Figure 1: System Configuration


Sensor data or Scenario generator
IKBS
library of space objects
objects in specified volume of space
Updates
data bank modified with time
Multiple Processor

transferred to a microVAX workstation in order to test the concept of the "High Performance Server" [1,2]. The idea of this concept is to enhance the host computer with add-on extensions. These extensions would be modular, based on a combination of any relevant architecture, and would produce a fast, high availability processor system for symbolic, numeric, graphic and conventional processing.

CURRENT WORK

The current phase of work includes orbital dynamics, fragmentation, object representation and sensor reasoning.

ORBITAL DYNAMICS

Calculations are being undertaken to assess the outcome of imparted impulses (ΔV's) to objects, fragments following explosions, collisions and hypervelocity impacts.

Modelling discrepancies were minimised by transfering from orbital elements (simple Keplerian motion) to cartesian co-ordinates [3,5,6] at the time a simulated event had been scheduled to take place. Resultant new velocities following fragementation were used to calculate individual debris/particle orbital elements.

An assumption was made that an operational space object described an unperturbed circular orbit. Following fragmentation, it was observed that the majority of resultant particles described elliptic motion; some achieved escape velocity. Simulations showed that the small particles (< 0.1gm) can have drastically different orbits from the initial parent body orbit. They had either very noticeable differences in orbital inclination and longitude of ascending node or large variations in semi-major axis and eccentricity. Resultant debris envelopes should correspond to those described by Fuss [4,7].

FRAGMENTATION

The number of fragments generated and fragment velocities following hypervelocity impacts and explosions were based on work by McKnight [11], Su & Kessler [14], and studies undertaken in the 1960's. The following assumptions were made in simulating fragmentation:

(i) Mass distribution of debris resulting from hypervelocity impact is a function of impact velocity and mass of projectile [9,11,14]
The ejecta mass is represented by

$$M_e = v^2 M_p$$

where M_e = ejecta mass in grams
v = impact velocity, km/sec
M_p = mass of projectile in grams

(ii) The ejecta mass distribution takes the form [9,11]

$$N = 0.447 (m/M_e)^{**}(-0.7496)$$

where N = number of fragments with mass m or greater
m = fragment mass
M_e = total ejecta mass

(iii) The general equations for debris resulting from the explosion of a satellite are [11,14]

$$N = \begin{cases} 1.71E\text{-}4\ M_t \exp(-0.02056\ m^{**}0.5)\ ;\ m > 1936\text{gm} \\ 8.69E\text{-}4\ M_t \exp(-0.05756\ m^{**}0.5)\ ;\ m < 1936\text{gm} \end{cases}$$

where N = cumulative number of fragments with mass greater than m grams
M_t = mass of target object in grams

(iv) The smallest detectable ejecta mass is of order 0.1gm

(v) The ejecta velocity for the smallest detectable fragment is 1.3 times impact velocity [9]. Velocities of larger ejecta particles were computed on the assumption that the kinetic energy was the same for all ejecta particles.

(vi) The maximum ΔV's imparted to fragments following an explosion are of order 2km/sec. Larger fragment velocities were calculated on the assumption that the kinetic energy was the same for all particles.

Initial results suggest that particle distributions could be detected showing the equivalent of a double shock wave following a hypervelocity impact and resultant explosion. Particle and velocity distributions may likewise indicate the orbital plane and velocity of the impacting body.

OBJECT REPRESENTATION

An object oriented approach was used to describe space object characteristics and relationships in terms of its attributes; namely instance variables and methods. Instance variables describe the simple variables or object relations that an object may possess; methods are operations that the object may perform.

A class of generic objects, termed operational component parts, was used to represent all functional/operational space objects such as satellites, rockets, spaceplanes and space stations. A sub-class of operational component parts was used to describe a typical space object. Terminology such as "forward-of", "behind", "above", "bottom", "left-of", "right-of" and "enclosed-by" was then introduced to represent spacially the component parts of the space object. In addition, each operational component part had assigned attribute fields describing mass, shape, material and structural properties. This is akin to the approach taken by such products as ICAD [13].

As an example consider the objects fuel tank, oxidiser tank, helium tank and rocket engines. These are a generic sub-group of operational component parts which can be used to represent the Space Shuttle's orbital maneouvering system [12]. These component parts could likewise represent the orbital control system for a space station or satellite.

Geometric reasoning was introduced to generate more accurate fragmentation models following impacts and explosions. The model assumed that debris was equally reflected and transmitted when the outer skin of a space object had been penetrated by an impacting body. It was further assumed that the transmitted debris damaged an operational component part causing the part to move into the class of non-operational objects. The model used rules to assess whether the damaged part was critical or non-critical; the outcome of this assessment resulted in debris being generated according to equations describing explosions or impacts respectively. Additional fracture and penetration models could be incorporated at this stage for more detailed simulations.

When a component part became non-operational following an impact or explosion, messages were passed to surrounding components to evaluate the outcome of this event. The outcome could be further explosions, shattering, fragmentation or fracture, resulting in adjacent components likewise moving into the class of non-operational objects.

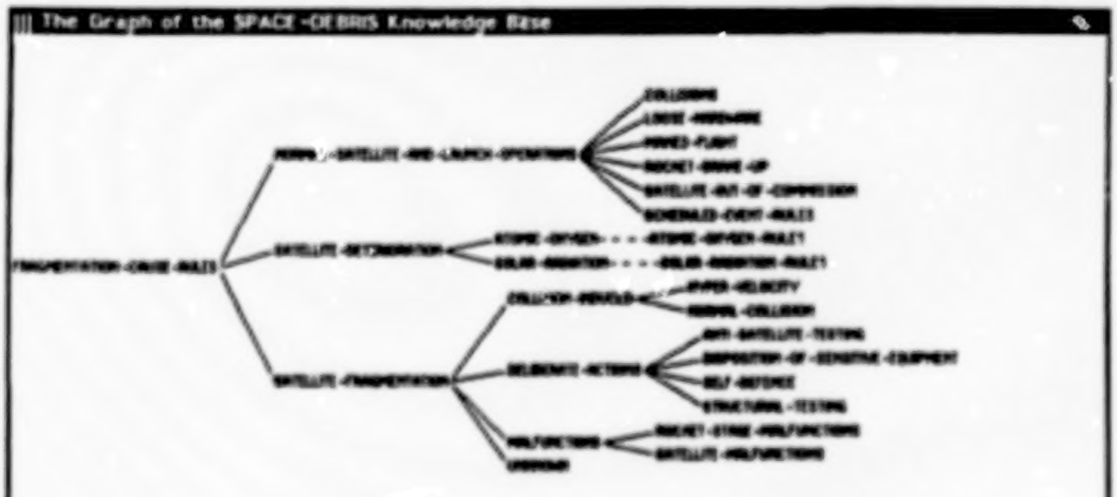
SENSOR REASONING

This was modelled as an intelligent system having inputs from various types of sensors and access to a library of currently known space objects. Consider the following scenario:

A possible cause for the loss of communication with a space object could be due to a collision, explosion or hypervelocity impact; this type of event being termed as a "non-scheduled event". The affected object or component parts of the affected object would transfer from the class of operational objects to non-operational objects. The sensor, having detected non-catalogued debris, would use fragmentation-cause-rules to determine the nature and origin of the detected debris, using as a key the time and known position of the suspected space object.

Fragmentation-cause-rules were classed under normal satellite-and-launch-operations, satellite-deterioration and satellite-fragmentation [8]. These were further categorized (see Fig 2) under collisions, loose-hardware, manned-flight, rocket-break-up, satellite-out-of-commission, scheduled-events, atomic-oxygen, solar-radiation, collision-induced, deliberate-action, malfunction and unknown.

Fig. 2



When non-catalogued debris was detected, the sensor reasoned as to the possible cause and origin. In undertaking this task, the sensor had access to an event library of non-scheduled and scheduled events. A non-scheduled event, as described earlier, could be a sudden loss in communication with an operational satellite. A scheduled event could be a hard-eject fragment following payload deployment into orbit. In this case, the following fragmentation-cause-rule was used to test this hypothesis:

```
HARD-EJECT-FRAGMENT
-------------------------------------
((IF (THE EVENT-TALLY OF ?DEBRIS IS ?EVENT)
     (?EVENT IS IN CLASS HARD-EJECT)
     (THE ACQUIRED-OBJECTS OF ?OBSERVATORY IS ?SIGHTING)
     (?DEBRIS = (LISP (FIRST ?SIGHTING)))
     (THE CATALOGUE OF ?OBSERVATORY IS ?CATALOGUE)
     (?FRAGMENT-CAUSE = (FOURTH ?CATALOGUE))
     (LISP (= ?EVENT  ?FRAGMENT-CAUSE))
     (?RCS = (LISP (SECOND ?SIGHTING)))
     (?RANGE = (LISP (THIRD ?SIGHTING)))
     (LISP
      (< ?RCS
         (OBJECT-MESSAGE 'NORMAL-ATMOSPHERE
                         '!ATTENUATE-SIGNATURE
                         '!RADAR-SIGNATURE
                         1.0e-4
                         ?RANGE
                         'KM)))
     THEN
     (THE DEBRIS-ORIGIN OF ?DEBRIS IS ?EVENT)
     (THE IDENTITY OF ?DEBRIS IS SMALL-FRAGMENT)))
```

If several fragmentation-cause-rules offered plausible solutions, each one could be tested concurrently. A possible confirmation of a rival hypothesis could be obtained by calculating orbital parameters at the time of the scheduled event and comparing the data with available fragmentation models.

CONCLUSIONS

The outcome of this study to date has shown that deep knowledge based simulations requiring symbolic, numeric and visualisation techniques could be linked together and applied to the artificial-debris problem. The use of KEE and SymTactics enabled rapid prototyping and provided an interactive and rapid scenario generation facility.

It is hoped that the outcome of this study will provide a better understanding of:

i) Object representation, damage assessment
ii) Intelligent sensor representation
iii) Object classification, discrimination and hazard/lethality.
iv) Generation of various space debris scenarios to provide a better understanding of the cause and break-up of operational space objects.

ACKNOWLEDGEMENTS

This work has been carried out with the support of the Procurement Executive, Ministry of Defence.

REFERENCES

1. Adamson, J.M., High Performance Server, Technical Paper, Thorn EMI Electronics Ltd., Oct '86.

2. Adamson, J.M. & Clay, P., High Performance Server Architecture, Paper in preparation.

3. Adamson, J.M., Determination of Relevant Satellite Data from Cartesian Co-ordinates or Orbital Elements, Technical Memo 2792, Space Division, Hawker Siddeley Dynamics, 1973.

4. Fuss, J.T., Dynamics of Explosion Remnants in Earth Orbit, Master's Thesis, Old Dominian University, July '74.

5. Gooding, R.H., Universal Procedures for Conversion of Orbital Elements to and from Position and Velocity (Unperturbed Orbits), Technical Report 87043, RAE, Farnborough, Jun '87.

6. Gooding, R.H., Solution of the Hyperbolic Kepler's Equation, Technical Report 87042, RAE Farnborough, Jun '87.

7. Humes, D.H. et al, Man Made Debris Studies at NASA Langley, Orbital Debris, NASA Conference Publication 2360, pp45-68, 1982.

8. Johnson, N.L. & McKnight, D.S., Artificial Space Debris, Orbit Book Company, 1987.

9. Johnson, N.L., History and Consequences of On-Orbit Breakups, Adv. Space Res., Vol.5, No.2,pp11-19, 1985.

10. Lorenz, R.D., Debris Threat Poses Future Hazard, Spaceflight, Vol 30, No.1,pp4-9, Jan '88.

11. McKnight, D., Determining the Cause of Satellite Fragementation: A Case Study of the Kosmos 1275 Breakup, IAA-87-574, Brighton.

12. Rockwell International Space Shuttle Transportation System - Press Information, Jan '84.

13. Rosenfeld, L.W. & Belzer, A.P., Breaking Through the Complexity Barrier: A New Style of Parametric Design, Technical Publication, ICAD Inc., Cambridge, MA, 1986.

14. Su, S.Y. & Kessler, D.J., Contribution of Explosion and Future Collision Fragments to the Orbital Debris Environment, Advances in Space Exploration, Vol 5, No.2, pp 25-35, 1985.

15. Wheatley, M., Marshall, G., & Magaldi, R., Configuration of Beta and Subsequent Releases of SymTactics : Pre-delivery Description, Ref VFC/1/88, Vanilla Flavor Company, Feb '88.

N89-15573

Extending the Data Dictionary for Data/Knowledge Management

Cecile L. Hydrick
and
Dr. Sara J. Graves
Computer Science Department
University of Alabama in Huntsville

Abstract

Current relational database technology provides the means for effeciently storing and retrieving large amounts of data. By combining techniques learned from the field of artificial intelligence with this technology, it is possible to expand the capabilites of such systems. This paper suggests using the expanded domain concept, an object-oriented organization, and the storing of knowledge rules within the relational database as a solution to the unique problems associated with CAD/CAM and engineering data.

I. Introduction

Data management for NASA often involves large amounts of diverse data stored on many different computer systems and at many different geographical locations. Types of data which must be tracked include project management data, financial and budgetary data, CAD/CAM data, engineering data, and documents. The possibility of using a single relational database management system (DBMS) for data connectivity has been explored. However, CAD/CAM and engineering data present problems which are not being currently addressed by existing DBMS products.

Although CAD/CAM and engineering data have the same basic requirements for storage and retrieval, certain characteristics of the data show why existing DBMS's fail. Such data (1) tends to be heterogeneous, consisting of graphical, textual, procedural, and mathematical data; (2) requires a dynamic schema as entities are created and destroyed; (3) tends to be object-oriented with complex relationships associated with the objects; and (4) exhibits object-specific relationships which change from object to object [5].

These characteristics require that a database designed for such applications (1) be able to represent a wide range of data types, (2) be able to represent complex relationships between data items, (3) and be able to represent certain "knowledge" about that data [2]. Existing commercial DBMS's do not at the present time have those capabilities.

The traditional data dictionary/directory may provide the answer. The data dictionary contains the "meta-data" which is the description of the data in the database. By extending the descriptions using knowledge representation techniques from the field of artificial intelligence (AI), the dictionary can in effect become the "knowledge base" for the DBMS, providing both dynamic schema generation and extended data types.

This paper presents methods for extending the data dictionary in a relational database management system by extending the domain concept to allow the representation of many different data types. Using an object-oriented model allows the expression of complex relationships between objects. "Knowledge" can be stored in the form of production rules mapped into a relational table. Combining the extension of the domain, the object-oriented model, and the storage of production rules in relational tables produces a data dictionary that is dynamic and capable of evolving over time, thus meeting the needs of CAD/CAM and engineering database applications.

PRECEDING PAGE BLANK NOT FILMED

II. Problems with Existing Data Dictionaries

The traditional process of database development resulted in a collection of static record structures which remained fixed throughout the life of the database applications. Data base administrators typically regarded data dictionaries/directories as static tools to aid them in the control of information resources [5].

The advent of CAD/CAM and engineering database systems has created the need for data dictionary definition to occur throughout the life of the application as objects are created, modified, and destroyed. The data dictionary, if viewed as a "knowledge base" rather than a collection of static records, can play an active role in this process. Expert knowledge about database design can be stored in the data dictionary itself, thus allowing for the creation of schemas as data loading occurs.

The key concept in the above scenario is that meta-data inherently contains knowledge which can be exploited for dynamic schema generation and knowledge management purposes. However, this will require that future systems be more tightly integrated than at present. In order to take full advantage of the knowledge inherent in the meta-data, data and meta-data can no longer be functionally separated, but must be made co-resident in the same "knowledge base". In this approach, database instances, data types, operations, and transactions are viewed as "objects". Two issues must be addressed in designing a knowledge-based data dictionary: a scheme for knowledge representation and the integration of the data and meta-data [5].

III. Moving from Data Management Towards Knowledge Management

Current DBMS's are effective in storing and retrieving large amounts of data. However, while a typical data dictionary may describe the physical size of the attribute "employee number", it may have no way to represent the fact that EMPLOYEE is a subtype of PERSON [11].

Artificial Intelligence research has produced Knowledge Representation Systems (KRS) which attempt to model the way in which human knowledge is represented and acquired. However, these systems have not been able to efficiently exploit large amounts of data due to the fact that they tend to be memory-based rather than disk-based [2]. Because they also tend to have high overhead and have ignored the issues of backup and error recovery, they have not been considered practical for large commercial applications [11].

Recent research has centered around finding ways to combine the best of both the DBMS and the KRS. There have been four approaches to integrating the two systems:

(1) integrate an existing AI system with a DBMS;

(2) enhance an AI system with data functionality;

(3) tightly integrate AI and database by designing an entirely new system;

(4) extend a DBMS by enhancing the data model with knowledge representation and other AI capabilities [2].

The fourth alternative forms the basis for this paper. This approach actually involves mapping knowledge into the existing DBMS. Several techniques have been used including the assorted semantic data models and the mapping of production rules into relational tables. An integrated approach must combine modelling richness with knowledge rules for inferencing capability.

IV. Expanding Domains in the Relational Model

Wedekind [10] discusses the importance of the domain in the design of conceptual schemata. He argues that the design process should reflect a learning situation in which single elementary facts about data are combined to form more complex knowledge about the data. His approach is a constructive method in which global domains are used as a basis for building relations in a stepwise fashion.

For example, a domain description such as COLOR = {green, red,blue} might be replaced with the following elementary sentences:

COLOR is a DOMAIN;
green is a COLOR.
red is a COLOR.
blue is a COLOR;

The fact that COLOR is a domain must be established before the members of the domain can be enumerated. In the same manner, the members of the domain must be known before an entity can be described as having that attribute.

In order to implement this concept using the relational model, Wedekind suggests the addition of four relations to the metadatabase: DOMAIN, ENUM (for enumerated types), and PRE and FUN which are relations specific to INGRES which allow the use of INGRES predicates and the calling of functions for validity testing.

Expanding the domain concept in engineering and CAD/CAM databases can provide two distinct advantages: (1) abstract data typing, and (2) "built-in" integrity constraints. The ability to represent abstract data types is necessary because of the heterogeneous nature of the data, while the enumeration of domain members provides the means of checking data entities for validity.

V. Object-Oriented Organization

One method for ascribing meaning to data is to describe the data in terms of objects rather than static record structures [5]. Objects can be entities or the relationships between the entities. Both declarative and procedural information can be included in the model [9].

Objects can be organized into "semantic nets". Primitives can include:

1) The class of an object of a certain type;

2) ISA-links which relate subtypes and supertypes;

3) ASA-links stating that a token is a specific type;

4) Primitive maps and functions which provide access to meta-data [5].

Both the object-oriented model and the relational database model share the goal of logical data independence, hence a natural mapping exists between the two models [9].

An object-oriented model that can be mapped into the relational model has been described by Sheldon Borkin [1]. His model is defined as the "semantic relation data model" and is based on the premise that the current database state consists of sets of statements describing the current state of the application. These statements are built from the meanings of natural language sentences which can be expressed as a verb phrase (predicate) plus several noun phrases.

An example of how an object might be mapped to the relational model is shown below. The object in Figure 1 has been named "ARCH" and consists of three elementary items named "BEAM1", "BEAM2", and "BEAM3". The functional decomposition of this object is shown by the graph in Figure 2. The relations resulting from the graph are shown in Table 1. It is interesting to note that the relations in Table 1 look similar to the way in which PROLOG states facts – i.e., supports(object1,object2).

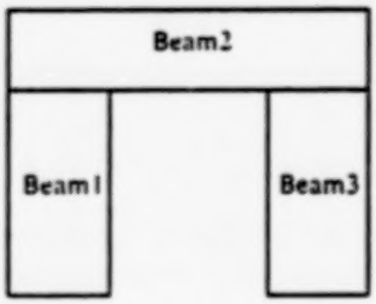


Figure 1: Object Named "Arch"

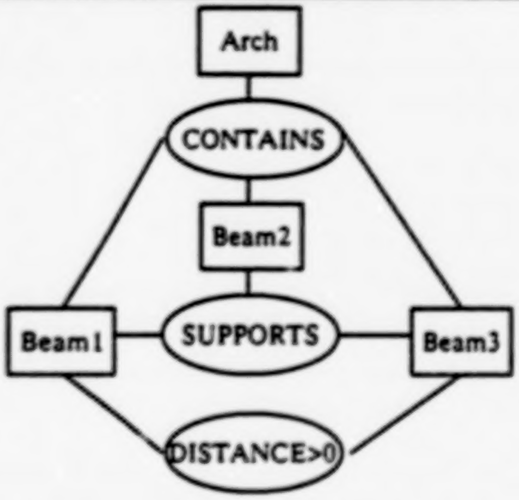


Figure 2: Functional decomposition

The object-oriented model has three advantages over the hierarchical, network, and relational models. First, the database can be viewed as a collection of abstract objects, not simply a group of two-dimensional tables. Second, both abstraction (attribute interconnection) and generalization (subtyping) can be more easily represented. Finally, object-oriented schema provide built-in integrity constraints [8]

Table 1: Relations for Object "Arch"

1) CONTAINS(Object1, Object2)

Object1	Object2
Arch1	Beam1
Arch1	Beam2
Arch1	Beam3

2) SUPPORTS(Object1, Object2)

Object1	Object2
Beam1	Beam2
Beam3	Beam2

3) DISTANCE>0(Object1, Object2)

Object1	Object2
Beam1	Beam3

As with most other existing models, trade-offs exist. Potential problems which may occur when using an object-oriented data model are added complexity and the difficulty in restructuring relationships once they have been defined [8].

VI. Storing Knowledge Rules for Inferencing

AI databases generally include two types of objects: facts about other objects and knowledge rules for inferencing [5]. In the data dictionary, facts can be represented using an object-oriented data model.

However, in order to maintain the close integration between data and meta-data, a method must be found for storing the knowledge rules within the database itself.

Recent work by Han-lin Li in China has centered on mapping production rules for inferencing into the relational model. This approach allows the relational DBMS to handle the matching and retrieval of production rules which have been mapped to the relational model.

Li's work centers around the basic form of the production rule which is "IF condition, THEN action with certainty factor CF." "AND" and "OR" operators are allowed to form complex conditions or actions. Table 2 shows some example production rules.

Table 2: Production Rules

Rule 1: If A = a and B = b and C = c THEN: D = d1 with CF(D) = cd1:

Rule 2: If A = a and B = b and C = c THEN: D = d2 with CF(D) = cd2:

Rule 3: If A = a and (B = b or C = c) THEN: D = d3 with CF(D) = cd3:

Rule 4: If A = a' or B = b' or C = c' THEN: D = d4 with CF(D) = cd4:

Table 3: Relations from Production Rules

R1, a relation containing rules 1 and 2

Rule#	IF:				THEN:		
	A	B	C	D	CF(D)	E	CF(E)
1	a	b	c	d1	cd1		
2	a	b	c	d2	cd2		

R2, a relation containing rules 3 and 4

Rule#	IF:			THEN:			
	A	B	C	D	CF(D)	E	CF(E)
3	a	b		d3	cd3		
	a		c	d3	cd3		
4	a'			d4	cd4		
		b'		d4	cd4		
			c'	d4	cd4		

The first two rules show that one condition may result in more than one action. Rule 3 shows that different conditions may result in the same action. Production rules with similar IF conditions form a relation.

The key for the rule relations is formed by combining the IF conditions for each tuple. This assures that the resulting relation will be in Fourth Normal Form. Table 3 shows how the production rules from Table 2 may be mapped into the relational model. Notice that in Table 2 the value of D must be included in the primary key because of the identical values of A,B,and C [9].

VII. A self-describing metaschema

Mark and Roussopoulos [10] have proposed an active and integrated data dictionary system which uses the services offered by the DBMS and is flexible enough to control its own evolution. They describe two orthogonal dimensions of data description: the point-of-view dimension and the intension-extension dimension.

The point-of-view dimension consists of three levels of data description: the external, conceptual, and internal schema. These three levels provide data independence.

The intension-extension dimension provides four levels of data description:

1) the application data;

2) the application schema which provides information about specific applications;

3) the data dictionary schema which provides information about the management and use of data;

4) the metaschema which consists of information about the data model.

Each level of description is the intension of the succeeding description and the extension of the preceding one. A description of the metaschema is explicitly stored as part of its own extension.

Using the object-role data model, the authors have mapped the core metaschema into object-oriented tables. The objects in a relational schemata are relations, domains, and attributes. The relation RELN defines relationships between existing relations and their names. ATTN defines relationships between attributes and their names. The relation DOMN describes the domains and the relation RDAS defines the relationships between relations, domains, and attributes. These mappings allow the metaschema to be stored in the database it defines.

The authors further describe a set of operations which control all operations on the data dictionary schema. They state that the operations specified must be explicitly represented in the metaschema itself in order for it to remain self-describing. Because object-oriented data models support storage of procedural information, this task is possible.

VIII. A Proposed Data Dictionary Architecture

In order to satisfy the need for dynamic structuring and closer integration between data and meta-data, it appears that both rules and facts must be actually stored in the database. With this in mind, facts about the data can be stored in relations modelled using the "semantic relation data model" and rules can be stored according to Li's mapping. These two representations match the way in which PROLOG defines predicates.

The core metaschema is designed using the base tables described by Mark and Roussopoulos. These tables have been enhanced by expanding the domain concept as described in section III. Figure 3 is the graph generated by combining these two concepts.

The circles in Figure 3 represent domains. The single boxes represent the actual relations, while the divided boxes represent the attributes associated with each relation.

The resulting relations are shown in Table 4. The relation DOMAIN has been extended to include information necessary for defining attributes in ORACLE, which was the relational product used in developing the prototype. Two additional relations further define the domain: (1) ENU allows enumerated domains for abstract data types such as objects, and (2) FUN stores the name of processes for testing and manipulation of the data. A final core relation RULES stores design rules for inferencing.

Other relations are created or dropped as the associated objects are added to or deleted from the database. When this occurs, the relations RELN, ATTN, and RDAS must be updated, thus ensuring a dynamic metaschema. The following example shows how an object might be added to the database.

In order to add the arch described in Section IV, the user would be asked whether the object was elementary or a composite of other objects. If the object is a composite, its components must be first added and described and discussed in Section III. For example, "BEAM1", "BEAM2", and "BEAM3" would be considered elementary items and must be stored in the database before the arch can be defined.

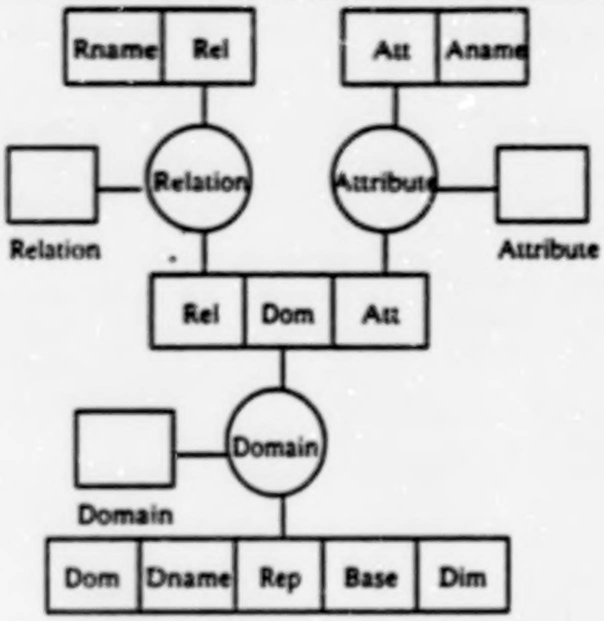


Figure 3: Prototype Core Metaschema

Table 4: Core Relations for Prototype

1)RELN(rname.	rel)
RELN	r1
DOMN	r2
:	:

2)DOMN(dom,	dname,	rep,	base,	dim)
d1	relation	STA	CHAR(c)	2
d2	attribute	STA	CHAR(c)	2
d3	domain	STA	CHAR(c)	2
:	:	:	:	:

3)RDAS(reln,	dom,	att
r1	d4	a1
r1	d1	a2
:	:	:

4)ATTN(att,	aname)
a1	rname
a2	rel
:	:

5)ENU(dom,	member)
d8	arch
d8	beam1
:	:

6)FUN(dom,	process)
d11	TSTAGE
:	:

7)RULES(cond1,	cond2,	cond3,	result)
:	:	:	:

The three beams are particular instances of the type beam which is a member of the domain object. A relation OBJECTS can be created to store these ASA links. The resulting relation is shown in Table 5.

Once the objects are added to the database, the relationships between them can be added. The two relationships SUPPORTS and DISTANCE>0 form two relations as shown in Table 5.

A design rule completes the process of describing the arch:

If SUPPORTS(x,z) and SUPPORTS(y,z) and DISTANCE>0(x,y)
  then ARCH(x,y,z);

Each condition evaluates to true if the specified tuple is found to exist. Thus application of this rule to the relations in Table 5 would confirm the fact that the object formed by the three beams is indeed an arch. The arch can be named and added to the OBJECTS relation and the relation CONTAINS can be added to describe the components of the arch. The resulting relations are shown in Table 6.

Table 5: Relations after Adding Beams

1) OBJECTS(oname,		otype)
	Beam1	beam
	Beam2	beam
	Beam3	beam
2) SUPPORTS(oname1,		oname2)
	Beam1	Beam2
	Beam3	Beam2
3) DISTANCE>0(oname1		oname2)
	Beam1	Beam3

Table 6: Relations after Design Rule

1) OBJECTS	(oname,	otype)
	Beam1	Beam
	Beam2	Beam
	Beam3	Beam
	Arch1	Arch
2) CONTAINS	(oname1	oname2)
	Arch1	Beam1
	Arch2	Beam2
	Arch3	Beam3

IX. Conclusion

CAD/CAM and engineering applications have special needs which are not presently being met by commercial DBMS's. These needs include the ability to represent abstract data types, relationships between the data items, and certain knowledge about the data.

Extension of traditional data dictionaries may be able to meet some of these needs. Extending the concept of domains allows the expression of abstract data types. Storing knowledge rules within the DBMS and using an object-oriented data model allow the representation of complex relationships.

Application of artificial intelligence techniques will allow the data dictionary to become a knowledge manager rather than a data manager. Only then will CAD/CAM and engineering databases be truly effective.

X. References

1. S. Cammarata and M. Melkanoff, "An Interactive Data Dictionary Facility for CAD/CAM Data Bases," Expert Database Systems: Proceedings from the First International Workshop, ed. Larry Kerschberg, Benjamin/Cummings, Reading, Mass., pp.423-440.

2. D.J. Hartzband and F.J. Maryanski, "Enhancing Knowledge Representation in Engineering Databases," Computer, Vol. 18, No. 9, Sept. 1985, pp. 39-48.

3. C.J. Date, An Introduction to Database Systems, Addison-Wesley, Reading, Mass., 1986, p. 39.

4. C. Zaniolo et al., "Object Oriented Database Systems and Knowledge Systems," Expert Database Systems: Proceedings from the First International Workshop, ed. Larry Kerschberg, Benjamin/Cummings, Reading, Mass., 1986, pp. 50-65.

5. J. Sowa, Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley, Reading, Mass., 1984, p. 304.

6. C. Kellogg, "From Data Management to Knowledge Management," Computer, Vol. 19, No. 1, Jan. 1986, pp. 75-84.

7. M. Brodie et al., "Knowledge Base Management Systems: Discussions from the Working Group," Expert Database Systems: Proceedings from the First International Workshop, ed. Larry Kerschberg, Benjamin/Cummings, Reading, Mass., 1986, pp. 19-33.

8. H. Wedekin, "Supporting the Design of Conceptual Schemata by Database Systems," International Conference on Data Engineering, Apr. 1984, pp. 434-438.

9. A. Shepherd and L. Kerschberg, "Constraint Management in Expert Database Systems," *Expert Database Systems: Proceedings from the First International Workshop*, ed.Larry Kerschberg, Benjamin/Cummings, Reading, Mass., 1986, pp. 309-331.

10. S. Borkin, *Data Models: A Semantic Approach for Database Systems*, MIT Press, Cambridge, Mass., 1980, pp. 63-93.

11. R. King, "A Database Management System Based on an Object-Oriented Model," *Expert Database Systems: Proceedings from the First International Workshop*, ed Larry Kerschberg, Benjamin/Cummings, Reading, Mass., 1986, pp. 443-468.

12. H. Li, "To Develop a Data-knowledge Base Management System by Utilizing Relational Database Management System", in Proceedings of Applications of Artificial Intelligence IV, (A Conference on 15-16 April 1986 in Inshruch, Austria).

13. M. Morgenstern, "The Role of Constraints in Databases, Expert Systems, and Knowledge Representation," *Expert Database Systems: Proceedings from the First International Workshop*, ed. Larry Kerschberg, Benjamin/Cummings, Reading, Mass., 1986, pp. 351-368.

14. L. Mark and N. Roussopoulos, "Metadata Management," Computer, Vol. 19, No. 12, Dec. 1986, pp. 26-36.

N89-15574

Case-Based Reasoning:
The Marriage of Knowledge Base and Data Base

Kirt Pulaski and Cyprian Casadaban

Martin Marietta Manned Space Systems
Post Office Box 29304 Mail Stop 3691
New Orleans, Louisiana 70189

Abstract

The coupling of data and knowledge has a synergistic effect when building an intelligent data base. The goal is to integrate the data and knowledge almost to the point of indistinguishability, therefore permitting them to be used interchangeably. Examples given in this paper suggest that Case-Based Reasoning is a more integrated way to link data and knowledge than pure rule-based reasoning.

1 Introduction

This paper describes some preliminary results of a NASA Mission Task being performed by the Automation & Intelligent Systems group at Martin Marietta Manned Space Systems in New Orleans, where the External Tank for the Space Shuttle is assembled. The goal of the project is to increase productivity at weld stations by decreasing downtime.

The plan to effect better productivity is to build an intelligent data base that gives advice about possible downtime causes and streamlines the follow-up paperwork. Efforts in the current fiscal year are producing a data base of reports of weld station downtimes as they occur. An ancillary knowledge base is growing as a result of the need to deepen the understanding of the weld station data.

A paradigm of reasoning needs to be selected that will best integrate the knowledge base and data base. Case-Based Reasoning (CBR) is being considered for several reasons:

PRECEDING PAGE BLANK NOT FILMED

CBR systems derive their power from knowledge base/data base interaction.

Both the knowledge and data are currently being collected case by case from the shop floor.

Advice-giving is memory-based and so is CBR.

Cases hold both data and knowledge in one structure, so the link between them is highly integrated.

CBR solves the problem of case disparity by being sensitive to and exploiting similarities.

The organization of this paper is to present the major components of the proposed Weld Intelligent Data Base (WIDB). First, a background of CBR is presented.

2 Case-Based Reasoning

The basic concept of CBR is simple: solve new problems by adapting solutions from old problems. The representation of a problem solving episode is called a **case**. Similarities between past and present cases establish a very high focus for problem solving. This focus is very difficult to achieve when only using rule-based reasoning (RBR).

CBR lends the power of examples to problem solving. Unlike the segmented explanations attributable to RBR, CBR explains its solutions with whole, relevant, concrete and familiar examples. CBR manifests learning as a by-product of adding cases, as they occur, to a case base. **Remindings** also provide a basis for knowledge acquisition (Riesbeck 88).

The generic system shown in Figure 1 depicts a cooperation between CBR and RBR (Pulaski 88). Each method has merits which the other lacks. RBR eases the implementation of a heuristic control strategy and is better for fast, non-complex, localized inferencing and for recording metaknowledge. CBR weaves the history of experience into problem solving.

3 Weld Data Base

When a downtime is reported a team of weld experts responds to the call. They work together to determine the cause of the problem, how to get the weld station operational as soon as possible and what to do to keep the problem from occurring again.

The result of a weld team call is a completed form which records, in several levels of detail, the path that the problem solving took from initial diagnosis through solution.

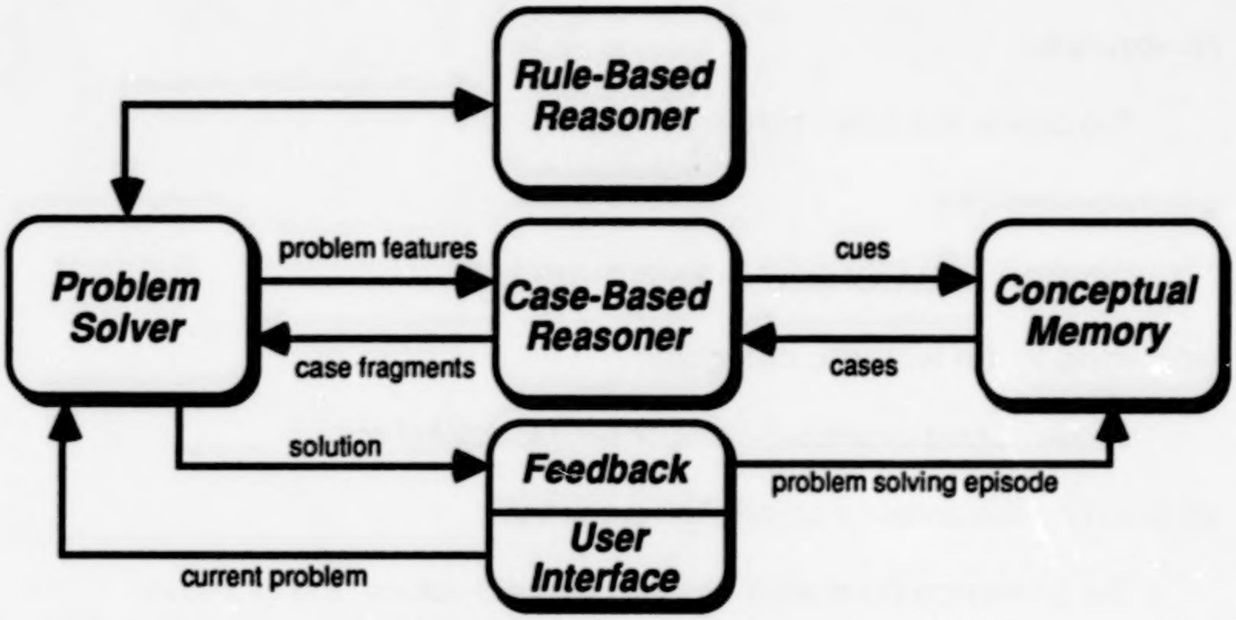


Figure 1: Generic Case-Based Reasoner

Each weld team report form is further broken down, analyzed and entered as a record into a data base. The year-to-date data base of weld station downtimes consists of about 250 records. Figure 2 shows an abbreviated version of a typical data base record that might result from a weld team call.

date	JAN-25-88
tool	T5018
effectivity	LWT-48
problem	TORCH-DOVE
cause	UNEVEN-TACK
downtime	H4
prepared-by	UNKNOWN
code	C88-21

Figure 2: Weld Team Report Data Base Record

4 Weld Knowledge Base

Figure 2 shows data from a weld team report. Certain types of knowledge need to be associated with the various fields, and the values in them, to enable an intelligent computational process to reason with that information.

One type of data base field knowledge is a systematic breakdown of allowable values for a given field. This knowledge is implemented as a **BNF grammar** which reduces the free English form to a parsed field value.

For example:

"No power to control system console."

is transformed into

CONTROL-SYS/CONSOLE/POWER/NOT/EXIST

according to the syntactic categories

System/Component/Subcomponent/Modal/Action

dictated by the problem description grammar.

The grammars do more than constrain field values. The grammar transformation rules can be used to support other functions such as form-filling and natural language (see Section 7). Also, the tokens of the grammars are often the same as the indices which cases in the case base are stored by, providing deep knowledge for storage and retrieval of cases.

Another type of knowledge relates field values to other field values. This knowledge is implemented as relational links in a field value hierarchy. Figure 3 shows part of the **Problem Tree**. The relational links are **ISA links**. For example, a PILOT-ARC-PROBLEM **ISA** TORCH-PROBLEM **ISA** PROBLEM.

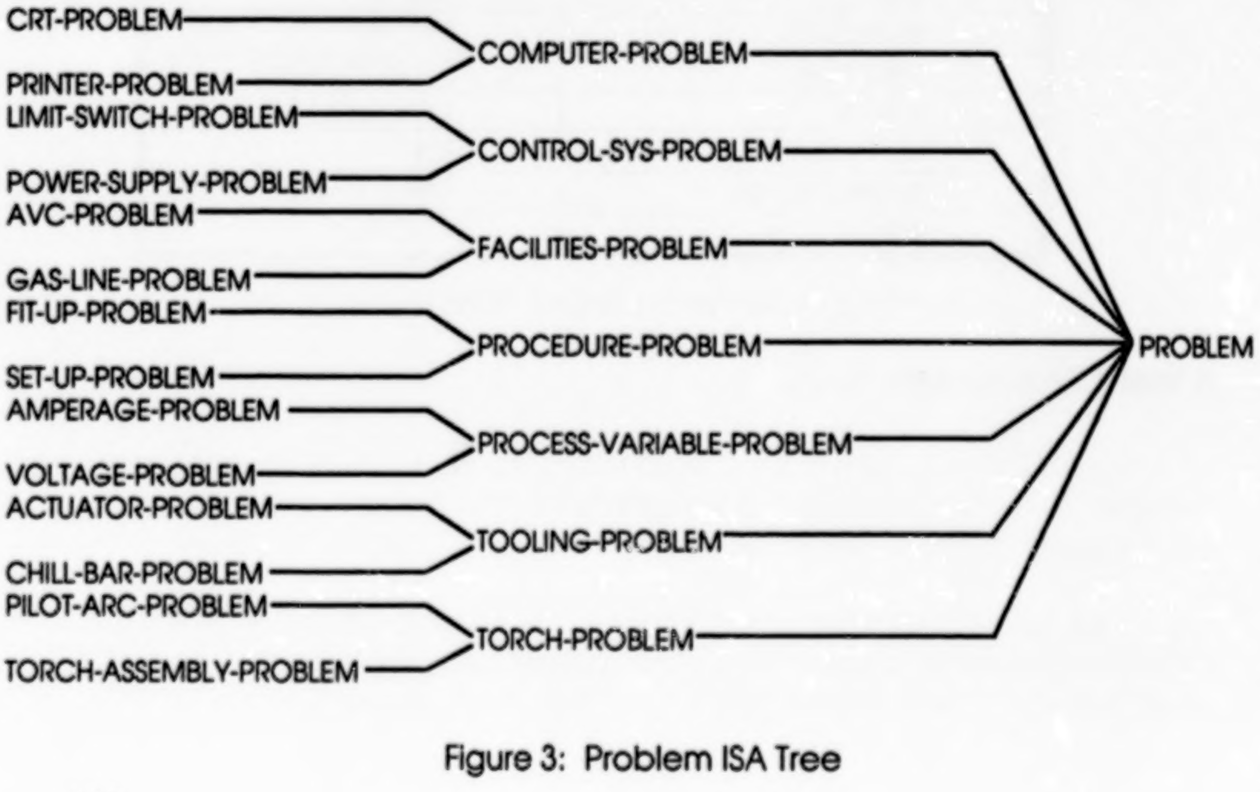


Figure 3: Problem ISA Tree

Relational knowledge helps to hypothesize uncertain or unknown knowledge in the data base. For example in Figure 2, the name of the person who prepared the weld team report is **UNKNOWN**. Using relational knowledge in the Problem Tree and the **Prepared-by Tree** helps to determine that the person who prepared the report was probably someone who usually handles **TORCH-PROBLEMs**.

Other relational knowledge can be used to link field values. For example, a **CAUSED-BY link** helps to consider causes for a problem. Any cause related to the current problem by a CAUSED-BY link is considered as a probable cause for that problem. Figure 4 shows that whenever there is a **TORCH-CUTTING** problem then three causes are considered.

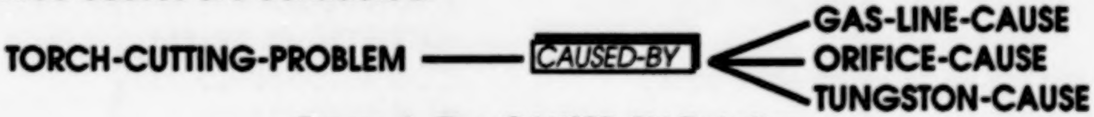


Figure 4: The CAUSED-BY Relation

5 Weld Case Base

The **weld case base** is a union of the weld data base and the weld knowledge base. The case base looks very similar to the data base but with an implicit deeper understanding of the values found in the cases. One source of the deeper understanding is the weld knowledge base. Another source is the knowledge that accumulates over time as reminded cases are analyzed and reasoned about during advice-giving.

Figure 5 shows an abbreviated version of a weld case from the case base. The case representation shows some of what CBR adds to problem solving. When **CASE-145** occurred, the CBR reasoner was **reminded-of** three previous cases which helped to postulate a set of **related-problems** to check for. Causes for the problem and related problems were hypothesized and investigated. Causes which were not substantiated were stored as **failed-causes** along with **reasons** why. The actual **cause** was substantiated and its **justifications** were recorded. Later, CASE-145 served as a **reminding-for** two other cases.

6 Weld Intelligent Data Base

The advisory function in the WIDB is directly supported by CBR. The design of the WIDB includes other functions that CBR does not directly support. The following paragraphs mention these functions and what approaches will be used to implement them.

The paper work currently associated with the data collection task will be automated. Human interface and form-filling methods will be used to accept user input, verify it, categorize it and build a case representation for the CBR advice-giver.

name	CASE-145
date	JAN-25-88
tool	T5018
effectivity	LWT-48
problem	TORCH-DOVE
down-time	H4
prepared-by	TORCH-EXPERT
code	C88-145
reminded-of	CASE-007 CASE-063 CASE-129
related-problems	AVC-PROBLEM WIRE-FEED-PROBLEM
failed-causes	CONTROL-SYS-CAUSE WIRE-FEED-CAUSE
reasons	VOLTAGE-OK WELD-BEAD-OK
cause	UNEVEN-TACK
justifications	MISMATCH-PROBLEM WELD-SEAM-PROBLEM START-UP-PROBLEM
reminding-for	CASE-158 CASE-240

Figure 5: Example from the Weld Case Base

Report generation using the data base will include summaries, statistical analysis, trending and relational analysis.

Relational analysis may be aided with the use of a knowledge discovery tool, for example **IXL by IntelligenceWare, Inc**.

The use of **similarity networks** (Bailey 88) is being considered to augment case generalization to improve relational retrieval. Better relational retrieval will improve advice-giving.

A natural language interface will ease querying, browsing and receiving advice.

7 Discussion

In an advice-giving domain it is very helpful to reason from past examples. CBR, by definition, is then a strong candidate for the WIDB domain, plus it has even more to offer. The work described here shows that CBR is especially useful when a solution involves an intelligent data base; that is, a data base cooperating with a knowledge base.

The performance of an intelligent data base depends on the level of integration between the data base and the knowledge base. CBR offers the highest level of integration since both data and knowledge are stored in cases and are not distinguishable.

The case base can grow in different ways. As more data and knowledge are acquired, more cases are built and added to the case base. Each occurrence of advice-giving is also a case, so the system grows each time it is used. More cases means better remindings for future advice-giving. This learning mechanism does not require the CBR reasoner to change; its performance increases as a result of better remindings. This caliber of learning is very difficult to achieve with pure rule-based systems.

There will be times when a CBR reasoner cannot solve a problem or subproblem. As Figure 1 suggests, a rule-based reasoner is then used to generate the unknown solution. This also leads to learning since the solution is saved in the case base and the rule-based reasoner never has to solve that problem (or other problems like it) again.

8 Conclusion

The wide-spread acceptance of a knowledge-based technique into a mainstream computing environment depends on several important issues: improvement, embedability and integration. These issues are important because knowledge-based techniques rarely produce an entire solution to a problem. Rather, the best solutions piece together a mix of subsolutions which use both knowledge-based and conventional approaches.

The design of the WIDB addresses these issues. First of all, a knowledge-based technique must offer an **improvement**. The WIDB augments a conventional data base with knowledge. The improvement is a better and deeper working understanding of the information in the data base; a result of linking data base field values to eachother with knowledge.

The WIDB uses CBR for advice-giving type problem solving. This offers an improvement since conventional methods have difficulty implementing advice-giving which is sensitive to a history that is constantly being modified.

Another issue is **embedability**: knowledge-based techniques must embed into the same environment as conventional methods. The knowledge that the WIDB uses to augment its data base is fully embedable into a relational data base environment. The knowledge links that connect data base field values to eachother are implemented as relations (for example CAUSED-BY and ISA) in the relational data base itself.

The cases for CBR are also embedable into a relational data base environment. CBR provides a seamless link between a data base and a knowledge base; in fact, the cases in the WIDB hold both data and knowledge. Therefore, the cases can be implemented as data base records.

The last issue is **Integration**. Different system components, whether knowledge-based or conventional, must fully integrate into one delivery environment. The WIDB integrates a CBR subsystem, a relational data base subsystem and a natural language front/back end. Each subsystem will be integrated with the others to operate in one delivery environment which is a 386-based microcomputer.

References

1. Bailey, D., Thompson, D., and Feinstein, J. "Similarity Networks." PC AI, July/August, 1988. pp. 29-32.

2. Pulaski, K. ELMO: An Episodic Long-term Memory Organizer for Case-Based Reasoning. AAAI Case-Based Reasoning Workshop, St. Paul, Minnesota, August 23, 1988.

3. Riesbeck, C. K. An Interface for Case-Based Knowledge Acquisition. In Proceedings of DARPA/ISTO Case-Based Reasoning Workshop, Clearwater Beach, Florida, May 11-13, 1988. pp. 312-326.

N89-15575

EXPERT SYSTEM VALIDATION IN PROLOG

Todd Stock, Rolf Stachowitz, Chin-Liang Chang, and Jacqueline Combs

Lockheed AI Center
Austin, Texas

ABSTRACT

We present an overview of the *Expert System Validation Assistant* (*EVA*) being implemented in Prolog at the Lockheed AI Center. Prolog was chosen to facilitate rapid prototyping of the structure and logic checkers and since February 1987, we have implemented code to check for irrelevance, subsumption, duplication, deadends, unreachability, and cycles. The architecture we have chosen is extremely flexible and expansible, yet concise and complementary with the normal interactive style of Prolog.

The foundation of the system is in the connection graph representation. Rules and facts are modeled as nodes in the graph and arcs indicate common patterns between rules. The basic activity of the validation system is then a traversal of the connection graph, searching for various patterns the system recognizes as erroneous. To aid in specifying these patterns, a metalanguage has been developed, providing the user with the basic facilities required to reason about the expert system. Using the metalanguage, the user can, for example, give the Prolog inference engine the goal of finding inconsistent conclusions among the rules, and Prolog will search the graph intantiations which can match the definition of inconsistency. Examples of code for some of the checkers will be provided and the algorithms explained.

Technical highlights include automatic construction of a connection graph, demonstration of the use of metalanguage, the A* algorithm modified to detect all unique cycles, general-purpose stacks in Prolog, and a general-purpose database browser with pattern completion.

N89-15576

EXPERT SYSTEM FOR ON-BOARD SATELLITE SCHEDULING & CONTROL

John M. Barry
Research & Technology Program Development Manager

Charisse Sary
Member of Technical Staff

SATELLITE & SPACE ELECTRONICS DIVISION
ROCKWELL INTERNATIONAL

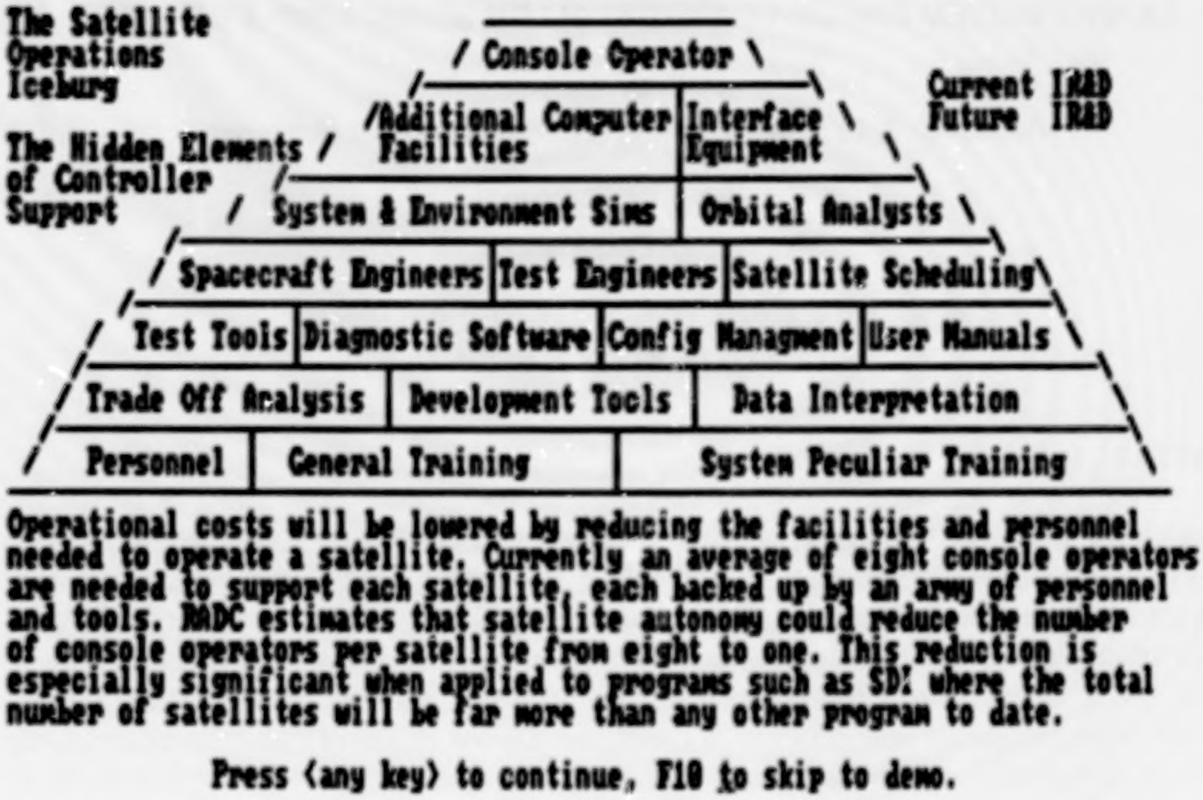
ABSTRACT

This paper describes an Expert System which Rockwell Satellite & Space Electronics Division (S&SED) is developing to dynamically schedule the allocation of on board satellite resources and activities. This expert system is the Satellite Controller. The resources it will schedule include power, propellant and recording tape. The activities controlled include scheduling satellite functions such as sensor checkout and operation. The scheduling of these resources and activities is presently a labor intensive and time consuming ground operations task. Developing a schedule requires extensive knowledge of the system and subsystem operations, operational constraints, and satellite design and configuration. This scheduling process requires highly trained experts anywhere from several hours to several weeks to accomplish. The process is done through "brute force" -that is examining cryptic mnemonic data "off line" to interpret the "health and status" of the satellite. Then schedules are formulated either as the result of practical operator experience or heuristics - that is "rules of thumb. Orbital operations must become more productive in the future to reduce life cycle costs and decrease dependance on ground control. This reduction is required to increase autonomy and survivability of future systems. The design of future satellites require that the scheduling function be transferred from ground to on board systems.

INTRODUCTION

Most satellite operations are accomplished by sending software commands via communications from ground control centers to the satellite. These commands monitor and control satellite "health and status" and uplink new schedules to control the satellite utilities and mission. The present method of managing these resources is through interpretation of digital satellite data, manually creating new schedules, then uplinking them to the satellite. This method, often referred to as the "software screwdriver," will dominate satellite operations until launch availabilities/capacities increase and launch costs are drastically reduced. Software techniques developed to manage these resources will supplement eventual on-orbit repair and replenishment schemes.

PRECEDING PAGE BLANK NOT FILMED



Operational costs will be lowered by reducing the facilities and personnel needed to operate a satellite. Currently an average of eight console operators are needed to support each satellite, each backed up by an army of personnel and tools. RADC estimates that satellite autonomy could reduce the number of console operators per satellite from eight to one. This reduction is especially significant when applied to programs such as SDI where the total number of satellites will be far more than any other program to date.

Press <any key> to continue, F10 to skip to demo.

Figure 1. The satellite operations iceburg

Even using the software screwdriver, operating and maintaining a satellite in orbit is a large, expensive, and complex task which requires many people, diverse skills, and coordination of various contractor and government organizations. Air Force studies indicate that an average of 8 controllers are required to operate and maintain 1 satellite. However, this figure is just the "tip of the iceberg". Backing up these controllers are "back room support" personnel such as orbital analysts, computer operators, programmers, systems engineers and so forth. This support easily expands into 200-300 people per satellite system (see Figure 1).

If we were to scale this present mode of support to the expected number of satellites for future space operations, the costs would be prohibitively high. We can no longer afford to control future spacecraft missions in the manner that we support such highly successful programs such as Viking and Voyager. Studies indicate that the satellite operations costs will rise dramatically if we continue these present methods. These increasing cost trends clearly indicate a need to simplify and automate the maintenance of satellites through an improved ground command and control environment. Ignoring these trends will severely limit NASA's ability to afford the acquisition, deployment and control of future space programs. Therefore, reducing ground command and control costs is a way to make more money available to develop future space programs.

The present method of analyzing and fixing problems, changing mission tasks on the ground, and sending commands back to the satellite must be changed. A loss of communication from the control centers due to war, terrorism or natural disaster would leave the satellite in a position where its mission might be degraded or unattainable. The design of future satellites require that the scheduling function be transferred from ground to on-board systems to increase autonomy, survivability, adaptability and reduce costs and response time.

TRADITIONAL APPROACHES TO SATELLITE CONTROL

Approaches to improve control satellites traditionally concentrate on automating computational and data reduction tasks, and developing better displays. However, these efforts alone will not solve the satellite Operational and Maintenance (O&M) problem. The solution is not trivial because significant engineering judgement and reasoning are required to operate the satellite and resolve anomalies. Satellite operation is complex because of the limited amount of on-board resources available such as electrical power. This situation is further complicated by multimission satellites which must share these resources among a variety of sensors. Sharing resources requires consideration of multiple constraints dependent on the sequencing of operations and availability of resources.

The management and planning of missions is presently accomplished by manually or automatically translating, sorting, and analyzing large amounts of digital data and displaying trends. A typical satellite console display contains only cryptic alphanumeric data that the operator must decipher. Some satellite operations centers then transmit this data to other computers for off-line analysis to display trend and graphical data. However, trend analysis is insufficient to accurately predict and correct all satellite anomalies. Such analysis cannot predict multidimensional, constraint-based anomalies or develop potential solutions to correct the anomalies.

INNOVATIVE ROCKWELL APPROACH

Rockwell Satellite & Space Electronics Division (S&SED) is developing an Expert System to dynamically schedule the allocation of on-board satellite resources and activities. This expert system is the Satellite Controller. The controller is a continuation of prior Rockwell on-board satellite intelligence research concepts. These concepts included not only the controller but also other "intelligent agents," such as the satellite planner, and subsystem specialist. The primary function of the planner is to generate a plan for fulfilling the objectives of a satellite or a group of mission related payloads. The subsystem specialist is responsible for the operational availability of its associated subsystem. The controller coordinates the generation of an agenda for executing selected missions of a satellite or group of mission related payloads. The controller is being prototyped to substantiate the concept of increasing on-board satellite autonomy. This concept also provides insights to simplify the task of the present satellite operations ground controller and the personnel who support ground control. This simplification reduces vast amounts of cryptic satellite data to create more intelligible operator displays.

The expert system of the satellite controller develops feasible strategies to manage the satellite resources and activities. These strategies are based on heuristics or "rules of thumb" currently used by ground satellite operations specialists. These heuristics are being incorporated into the reasoning algorithms of the Rockwell expert system. Rockwell will transition the ground expert system into future satellite designs once they have been proven and tested in the ground control environment.

The Rockwell approach is based on examining current design and operations of several satellite systems which it is currently designing, producing or operating. These systems include the various navigation and surveillance satellites. Our approach starts with a functional examination of the objectives needed to operate and maintain a satellite in the most cost effective manner. The Rockwell concept concentrates on presenting knowledge or formulating advice instead of displaying only raw information to an on-board controller. This knowledge is the result of known constraints, an operational model of the satellite systems, and the judgement developed by experts. Today this knowledge is created by the previously mentioned "back room support" personnel. The Rockwell approach is to reduce and display digital data in a manner which simplifies operator understanding. This approach will result in real time satellite control and analysis which can be implemented within the spacecraft systems to increase autonomy.

The innovative Rockwell approach described in this paper and demonstrated on a personal computer in this conference covers several facets. These facets include the Satellite Controller Concept, the Enhanced User Interface, the Knowledge Base, the Satellite Controller Description and Operation, and Mission Planning. This later facet will be illustrated to the user by leading him through a scenario handled by the Rockwell Satellite Controller.

RAPID PROTOTYPING

Rockwell developed a low risk, high confidence approach to the controller design through rapid prototyping. Rapid prototyping is a technique which one models the visual interface and operation, but not complete functionality of the desired product. The controller's perspective was obtained through dialogue and feedback from current Air Force satellite operations personnel. This perspective emphasized simplification of the user interface and reduction in the number of operational personnel. We used this information and rapid prototyping to encapsulate the satellite controller actions. The result was deemed by representatives of the Air Force to accurately reflect the visual cues a satellite controller would like to see. Designing this perspective allows us to simplify ground control mechanisms and functions and understand the processes required to design more autonomy into satellites. The prototype is designed so that it can be readily changed to reflect enhancements to the controller's perspective and true operation of the system without extensive recoding.

THE SATELLITE CONTROLLER CONCEPT

Controlling satellites is a labor intensive and time consuming ground operations task. Developing a schedule requires extensive knowledge of the system and subsystem operations, operational constraints, and satellite design and configuration. This development process requires highly trained experts anywhere from several hours to several weeks to accomplish. The process is done through "brute force" - that is, examining cryptic mnemonic data "off line" to interpret the "health and status" of the satellite. Then schedules are formulated either as the result of practical operator experience or heuristics - that is "rules of thumb." Rockwell is developing the Controller to improve orbital operations productivity, reduce life cycle costs, and decrease dependence on ground control.

The Rockwell Satellite Controller is an expert system which controls, coordinates, and manages the activities of various subsystem specialists. Subsystem specialists control and manage their respective subsystems such as the propulsion, power, attitude control, or communication subsystem. The coordination is achieved through an agenda or common area that either the controller or the subsystem specialists can access. Requests or statuses of actions are posted on the agenda. This information is used by the controller in creating an initial schedule and in coordinating its execution. This IR&D project has concentrated on developing the satellite controller and simulating the activities performed by the subsystem specialists. Also, this IR&D project has begun to determine the division of knowledge between the controller and the subsystem specialists. The controller is knowledgeable of the information necessary to make global decisions that may affect the subsystem specialists, whereas the subsystem specialists are knowledgeable of the information specific to their respective subsystem.

SCHEDULING LOGIC

The scheduling logic was developed in parallel with the prototype user interface using CLIPS, a C-based expert system building tool developed by NASA. Controlling this process involves an inference mechanism known as forward chaining. Forward chaining is an inductive mechanism which uses facts and rules to "reason" toward a solution. This mechanism examines the premises of rules to see whether or not they are true given the information on hand. If so, then the conclusions are added to the list of facts known to be true and the system examines the rules again. The satellite controller uses this process through the information in the knowledge base, concerning interpretive rules and information about the design and operation of the satellite systems.

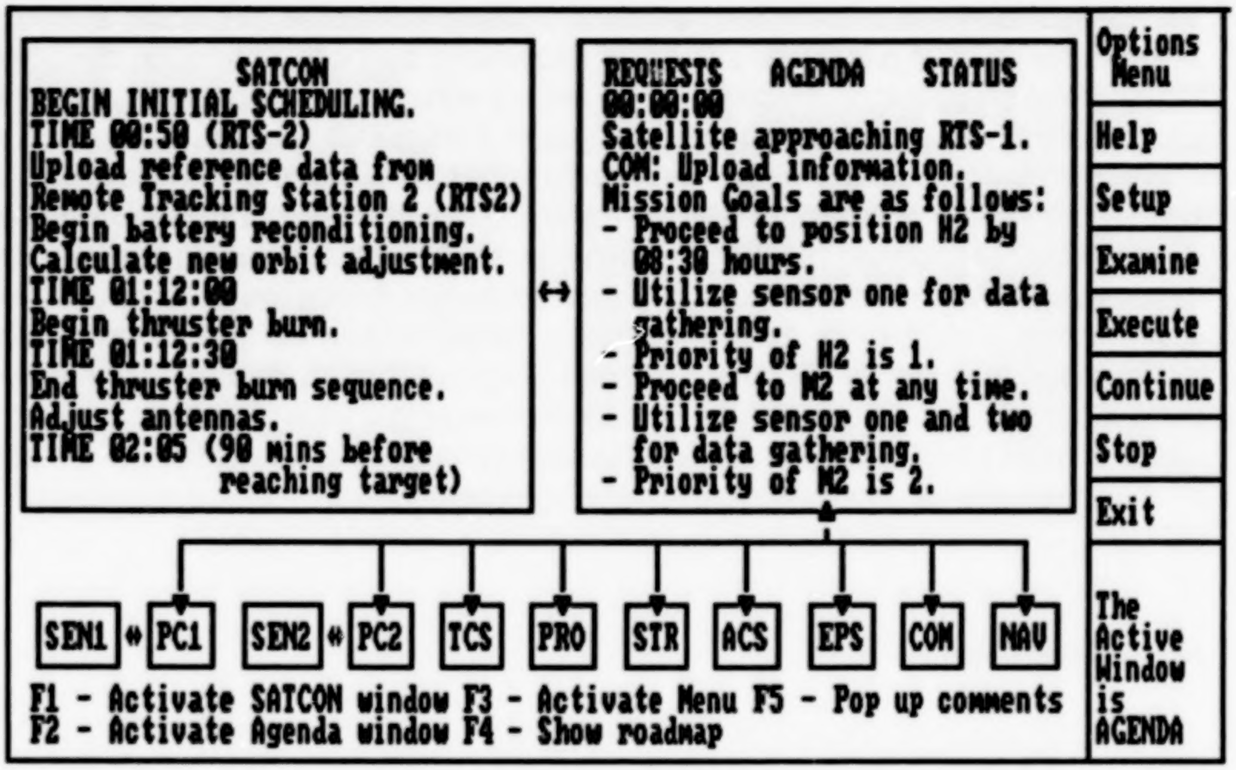
ENHANCED USER INTERFACE

The prototype of a user interface concentrates on displaying relevant knowledge - i.e. "digested information," meaningful to the operator. This interface can replace digital data from several operator terminals with a single screen displaying English language phrases which do not require deciphering. Therefore, the operator is presented with the phrase "sensor 1 slewed 5 degrees" instead of the normal digital data that must be interpreted. This process is more than a simple transformation. It actually involves parsing or interpreting inferred information from the inputs of several systems aboard the satellite.

The controller interface generates four key groups of data during its execution: agenda information, satellite controller actions, subsystem health and activity status, and task schedule timelines. The user interface was developed to demonstrate understanding of an on-board design approach, portray a potential ground station controller's workstation, and provide user control of the expert system simulation.

The English phrases are displayed in one of two windows of the Satellite Controller. The main screen is divided up into a SATellite CONtroller (SATCON) window, an Agenda window, Subsystem Icons and a MENU. The SATCON window displays the actions of the controller as it creates a schedule. The Agenda window shows REQUESTS for action from either a subsystem or the controller. The Agenda window also shows the current STATUS of the various subsystems of the satellite. These windows can be activated by a macro key on the terminal which toggles between the two activities. Both windows can be scrolled to display an audit trail of activities that have occurred on the satellite. The display has icons on the right side of the screen which activate other macros to allow the operator to "EXAMINE" the schedule, a "HELP" key, and others. In addition, the satellite subsystems are displayed in icons across the bottom of the screen. The individual subsystem icons are activated whenever activity is occurring which affects that subsystem. This activation assists the operator in visualizing subsystem status (see Figure 2).

The enhanced user interface of the satellite controller replaces the digital display mnemonics of current systems. More importantly it consolidates information of several satellite operators and support personnel into one display. The experience gained from this design will be used to define the data flows for the eventual on-board controller.



```
SATCON
BEGIN INITIAL SCHEDULING.
TIME 00:50 (RTS-2)
Upload reference data from
Remote Tracking Station 2 (RTS2)
Begin battery reconditioning.
Calculate new orbit adjustment.
TIME 01:12:00
Begin thruster burn.
TIME 01:12:30
End thruster burn sequence.
Adjust antennas.
TIME 02:05 (90 mins before
              reaching target)
```

```
REQUESTS   AGENDA   STATUS
00:00:00
Satellite approaching RTS-1.
COM: Upload information.
Mission Goals are as follows:
- Proceed to position H2 by
  08:30 hours.
- Utilize sensor one for data
  gathering.
- Priority of H2 is 1.
- Proceed to M2 at any time.
- Utilize sensor one and two
  for data gathering.
- Priority of M2 is 2.
```

```
F1 - Activate SATCON window F3 - Activate Menu F5 - Pop up comments
F2 - Activate Agenda window F4 - Show roadmap
```

```
The
Active
Window
is
AGENDA
```

Figure 2. The Satellite Controller User Interface

A real user interface to visualize the controller's perspective was coded in Turbo C to be used by the scheduler. This scheduler, plus the user interface and the knowledge base, provides the platform for the Satellite Controller expert system development.

KNOWLEDGE BASE

Construction of a knowledge base is a complex process which involves an intimate knowledge of the subsystems, their relationships, constraints/rules and operating parameters. Initially, the Rockwell knowledge base is purely rule based. Eventually we will organize this knowledge base into frames which is a knowledge representation scheme that associates an object with a collection of features (e.g. facts, rules, defaults, and active values). This knowledge representation facilitates the development of model based reasoning schemes. Model based reasoning can create dynamic schedules based on a system representation rather than pure rule based system. The advantages of a model based system is that it can infer an "unknown" situation not specifically stored in the knowledge base.

The contrast between a rule based and model based automobile diagnostic system illustrates this point. A rule based system can accurately diagnose a condition directly attributable to a procedure or checklist which is already in the knowledge base. For example, such a system can isolate a failed voltage regulator if the car will not start. However, if an unpredicted event such as a meteor fell on the engine the night before, the rule based expert system would not accurately diagnose the problem. However, a model based system contains not only rules but the description on "how" the system should operate. This description includes hierarchical or complex relationships among the systems and message passing. This description actually forms a "model" on how the system works. This model based reasoning would then determine that there is major damage to the engine compartment or some subsystem(s) instead of developing a false diagnosis. This information is more useful and accurate than that developed by a rule based system because it can make inferences on dynamically changing situations.

MISSION PLANNING

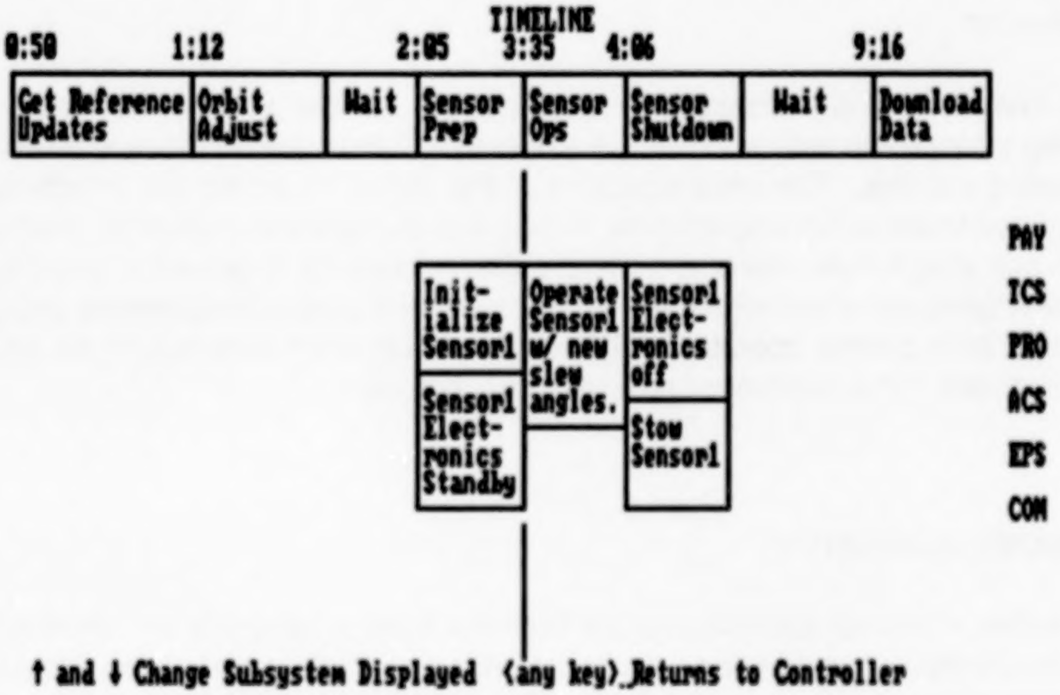
As a premise to scheduling activities and resources, the expert system performs mission planning. A mission scenario was created to validate these concepts. Initially the planning would be developed and tested a ground control workstation. Rockwell plans is investigating the use their Mission Operations Support Center to construct and test a satellite controller on its Global Positioning System. The ultimate goal is to develop the technology to design this expert system to operate on-board future satellites.

A mission consists of a goal or objective, a start time, a duration, and a priority. The present Rockwell expert system is modeled after a surveillance satellite. A typical mission might be to view a ground location at a prespecified time. A mission is made up of multiple tasks that must be completed in order to satisfy a mission. For a viewing mission, typical tasks that must be scheduled would include operating a sensor, preparing a sensor for operation, shutting down a sensor, and downloading data to a remote tracking station when recorders are full following a mission. General station keeping tasks must also be scheduled such as orbit adjustments, uploading current reference updates when over a remote tracking station, or momentum dumps.

Given multiple, conflicting missions, the expert system will try to schedule as many missions as possible. Currently, priority is the only constraint used to determine which missions will be scheduled first and which missions cannot be scheduled at all. The satellite moves on a path over the earth called a ground track and can move or slew itself several degrees in the plus or minus direction in order to view a location. Therefore, it would be possible to view two locations when on the same ground track by slewing the sensor. It could also move to another ground track to view a location, but this will require resources such as propulsion in order to make the move. In the future, the expert system will incorporate the reasoning to determine how to satisfy as many missions as possible by traveling on a ground track where multiple locations could be viewed at once while minimizing the amount of resources used.

CONSTRAINT BASED TASK SCHEDULING

Constraint based reasoning is used to assist in the mission planning function. Given several mission goals, the expert system will determine what tasks must be accomplished in order to satisfy a mission goal. It will determine when to schedule these tasks based on constraints. Typical constraints include temporal constraints such as prepare for sensor operation must be done before sensor operation or shutdown sensor must be done after sensor operation (see Figure 3). Other constraints include scheduling a download data task after all recorders are full. This can be calculated by summing the durations of sensor operation tasks. Or if it has been over 90 minutes since gyro heaters have been turned on then schedule a 90 minute prepare for sensor operation task in order to allocate enough time to run the gyro heaters before a sensor operation task begins. Tasks are made up of subtasks. For example, a sensor preparation task is made up of tasks to power up the payload and initialize it, turn a recorder to standby, turn the payload electronics to standby, and perform enhanced attitude adjustment. Enhanced attitude adjustment includes turning on the gyro heaters, enabling the GN2 thrusters, enabling the rate gyros, and maintaining attitude. Scheduling is performed at the task level if possible, otherwise scheduling can be done a any subsequent task level below.



↑ and ↓ Change Subsystem Displayed (any key) Returns to Controller

Figure 3. The Satellite Controller display for a sample sensor payload schedule

RESOURCE MANAGEMENT

Included in scheduling of tasks and subtasks is scheduling of the on-board resources which enable the task to be completed. Currently, the system will determine if a task can or cannot be scheduled based on available resources. In the future, it will be able to reason about when would be the best time to perform a task based on the resources the task will use. For example, it is better to perform a task on the current ground track, rather than move to another ground track because less propulsion will be used. Currently, three resources are managed: power, propellant, and recording tape. Power is a resource that stays at a fixed level and is reduced or increased when a task is performed or completed. All tasks use power and a minimum amount of power is always used for station keeping. Propellant starts at a given level and is used as tasks are performed but it is never replenished. Recording tape is used during a sensor operation and is completely used when all recorders are completely full of data. Recorders are replenished when all data has been dumped to the ground when over a remote tracking station.

These are just a few resources that must be considered when designing the Satellite Controller. Future research will concentrate on expanding the controller functionality to handle other missions. This research will be integrated with other Rockwell projects with reliability, fault detection and diagnosis.

SUMMARY

The Rockwell Satellite Controller project is using Artificial Intelligence technology to develop a concept to reduce future space operational costs and increase effectiveness in controlling satellites. The initial objectives of the Rockwell project are to schedule onboard satellite resources and activities. In the process, Rockwell is developing techniques which can simplify operations and improve the productivity of ground controllers. The Rockwell approach of investigation is based on examining system operations and gaining feedback from satellite operators. This feedback was used to construct the prototype demonstrated in this conference on a personal computer.

ACKNOWLEDGEMENTS

The author wishes to acknowledge the Rockwell team responsible for developing the Satellite Controller concept and building the prototype. This team is in the Software and Simulations Department of the Satellite and Space Electronics Division of Rockwell. Linas Raslavicius, the Project Coordinator, and Tom Gathmann, knowledge engineer are the original architects of the Controller Concept. Other members involved in the design and evolution of the concept and prototype include Jonathan Kass, Dave Mattox, Tom Gathmann, Sumalee Johnson, JoAnne Pitts and Billie Shannon. Members of the team also provided valuable editorial comments which are reflected in this paper. The author also thanks the many people from the Rome Air Development Center, AF Space Command, AF Space Division and the Aerospace Corporation for their encouragement and comments throughout this project.

BIBLIOGRAPHY

Harmon, P. and D. King. Artificial Intelligence in Business. New York: John Wiley, 1985.

Harmon, P. and D. King. Expert Systems: Tools and Applications. New York: John Wiley, 1987.

Hayes-Roth, F., D.B. Lenat, an D.A. Waterman (Eds.) Building Expert Systems. Reading, Ma: Addison-Wesley, 1983.

BIOGRAPHY

John Barry is currently the Program Development Manager for Research & Technology at Rockwell's Satellite & Space Electronics Division in Seal Beach, California. Prior to joining Rockwell he was an Air Force Officer for 20 years serving the USAF Space Division and Rome Air Development Center. He has a B.S. Degree in Math and Physics, from LeMoyne College, an M.S. Systems Management from the University of Southern California, and M.S. in Logistics Management from the Air Force Institute of Technology and an M.S. in Computer Science from Northrop University. Mr. Barry is an adjunct professor in the graduate program at Northrop University in the Computer Science Department. He is also a Senior Member of the Society of Logistics Engineers.

Charisse Sary is a Member of the Technical Staff and Technical Lead for the Satellite Controller in the Software and Simulations Department at the Space & Electronics Division. She has an M.S.E. in Computer Science and Computer and Information Science with a specialization in Artificial Intelligence from the University of Pennsylvania. She has over 9 years experience in computer science with 3 years experience in Artificial Intelligence and expert systems at Rockwell.

N89-15577

Dypas:
A Dynamic Payload Scheduler for Shuttle Missions

Presented by Stephen Davis
Johnson Research Center
The University of Alabama in Huntsville
Research Institute A-11
Huntsville, Alabama 35899
(205) 895-6257

Decision and analysis systems have had broad and very practical application areas in the human decision making process. These software systems range from the 'help' sections in simple accounting packages, to the more complex computer configuration programs. Dypas is a decision an analysis system that aids the scheduling process done before a shuttle mission, and has added functionality to aid the rescheduling process done in flight. Dypas is written in Common Lisp on a Symbolics Lisp machine. Dypas differs from other scheduling programs in that it can draw its knowledge from different rule bases and apply them to different rule interpretation schemes. The system has been coded with Flavors, an object oriented extension to Common Lisp on the Symbolics hardware. This allowed the implementation of objects (experiments) to better match the problem definition, and allow a more coherent solution space to be developed.

Dypas was originally developed to test a programmer's aptitude toward Common Lisp and the Symbolics software environment. Since then the systems has grown into a large software effort with several programmers and researchists thrown into the effort. Dypas is currently using two expert systems and three inferencing procedures to generate a many object schedule. the paper will review the abilities of Dypas and comment on its functionality.

PRECEDING PAGE BLANK NOT FILMED

N89 - 15578

A Knowledge-based Decision Support System for Payload Scheduling

Rajesh Tyagi
Fan T. Tseng
Department of MIS/MSC
University of Alabama in Huntsville
Huntsville, AL 35899
(205) [illegible]95-6510

ABSTRACT

This paper presents the development of a prototype Knowledge based Decision Support System, currently under development, for scheduling payloads/experiments on space station missions. The DSS is being built on Symbolics, a lisp machine, using KEE, a commercial knowledge engineering tool.

INTRODUCTION

The task of payload scheduling is rather complicated. It not only involves using algorithms to generate a schedule, it is also very dynamic in nature since the schedule frequently needs to be revised, often at short notices, due to unpredictable and uncontrollable features like equipment failure, power failure, and other emergencies involving a mission.

The traditional approach taken in solving such problems is to develop a decision support system using conventional programming tools. The purpose of this research is to use artificial intelligence/expert system techniques, both hardware and software, to develop a knowledge-based decision support system for this space station scheduling problem.

The DSS should be able to generate a payload/experiments schedule based on the needs and the objectives of the user. It should also be able to update/modify the schedule as those needs and objectives change over the course of the mission.

THE DSS

The knowledge-based DSS is comprised of three components as shown in Figure 1. These components are: a knowledge base, a models base, and a user interface. The knowledge base possesses information on various attributes of the experiments (like power and experiment run time), and the available resources (like power supply). The models base contains analytical and heuristic models that may be used to develop the experiment schedule. And the user interface provides the dialog between the user and the knowledge and models base.

The Knowledge Base

The knowledge base is organized in the form of frames. Each experiment is represented by a frame, the slots of the frame representing different attributes of the experiment. Figure 2 shows the frame corresponding to experiment 'Crystal Growth". The experiment, which is sponsored by NASA, is to be run only once, the run requiring a power supply of 500 kilowatts over 50 hours, the duration of the experiment. This data is inputted by the user before the schedule is generated. The starting and ending times for the experiment are determined by the analytical model used to generate the schedules, and are automatically placed in their respective slots. The resources are also similarly represented by frames in the

PRECEDING PAGE BLANK NOT FILMED

knowledge base, one frame for each resource. Figure 3 illustrates the overall structure of the knowledge base.

FIGURE 1.

THE DSS ARCHITECTURE

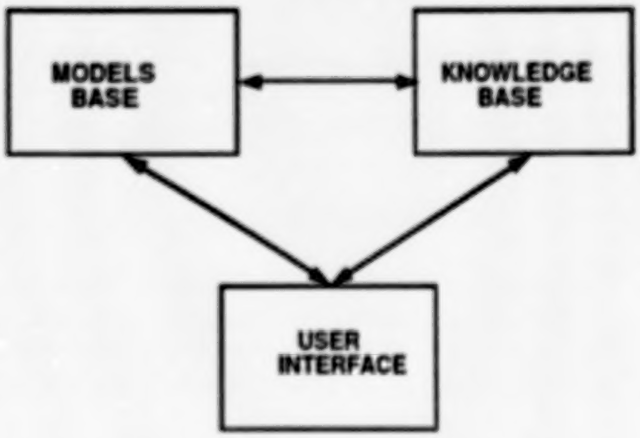


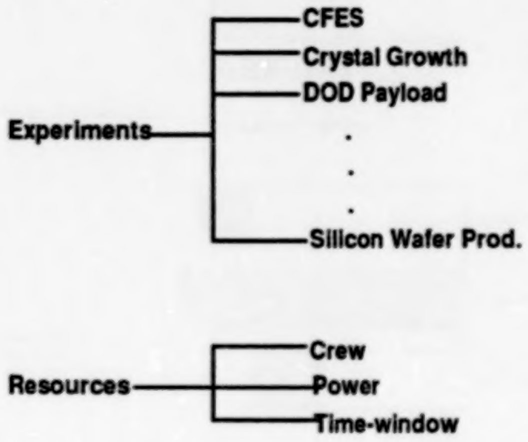
FIGURE 2

Frame for Crystal Growth

SLOT	VALUE
Agency	NASA
Duration	20 hrs
Power	1200 kw
Runs	1
Starting Time	41
Ending Time	60

FIGURE 3.

THE KNOWLEDGE BASE



The Models Base

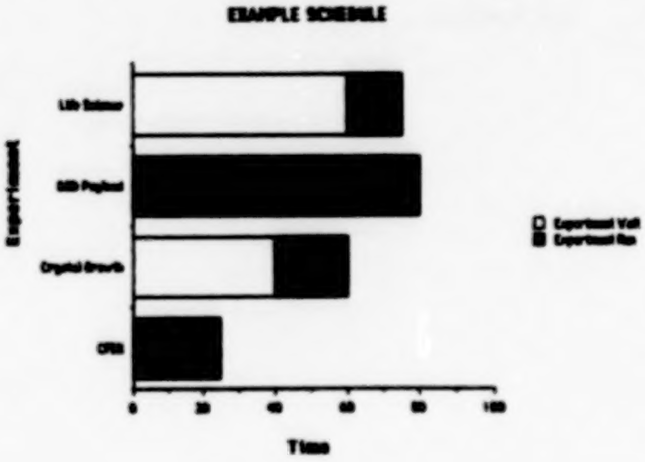
The models base contains a set of analytical models that may be used to determine a schedule, based on the objectives and/or requirements of the user. The user may specify the criteria that is to be used for scheduling. These criteria include: (1) minimizing the total time needed to schedule a given set of experiments, and (2) leveling the power consumed given that a set of experiments are to be scheduled within a specified amount of time.

As mentioned earlier, the space station scheduling problem is very complicated, given the set of constraints imposed by the experiments. Most of the current literature deals with developing heuristics to generate the schedule, since the schedules need to be generated in relatively short amount of time, often at short notices, which makes it impractical to use optimizing algorithms which often require a lot of execution time to generate a solution. However, this may not always be a good approach to take since time needed to generate a schedule is not always of the highest concern. In fact, the scheduling task may be broken down into two different phases: (1) constructing an initial schedule, and (2) dynamically modifying the schedule during the mission as needed due to equipment failures etc. The initial schedule, whose planning horizon is spread over a period of weeks/months, could be generated using optimizing algorithms since execution time is not a major concern during this phase. The disadvantage of longer processing times of optimizing algorithms is outweighed by the payoff from a far better schedule that may be generated. It is only during the second phase, when schedules must be modified expediently, that heuristics may be preferable to optimizing algorithms. Thus it is desirable to have a full spectrum of scheduling models ranging from highly optimizing algorithms to 'quick and dirty' heuristics.

The User Interface

The user interface provides the dialog between the user and the DSS in the form of windows, menus, and graphical displays. The user controls the execution of the system by specifying the criteria to be used in determining the schedule, and using menus to update the knowledge base, and updating/modifying the schedule. The output of the DSS is in the form of updated knowledge base and/or graphical displays of the schedule generated. Figure 4 shows an example schedule.

FIGURE 4



CONCLUSION

This paper has presented an overview of a knowledge-based DSS, currently under development, for experiment scheduling on space station missions. The DSS utilizes artificial intelligence/expert system techniques to build the knowledge base and the user interface.

REFERENCES

1. Baker, K. R., Introduction to Sequencing and Scheduling, New York: John Wiley and Sons, Inc. 1974.
2. KEE User's Manual, Intellicorp Inc.
3. Lawler, E. L., Lenstra J. K., and Kan, A. H. G. Rinooy, *"Recent Developments in Deterministic Sequencing ans Scheduling: A Survey,"* in M. A. H. Dempster et. al. (Eds.), Deterministic and Stochastic Scheduling, Reidel, Dordrecht, 1982, 35-72.
4. Symbolics software. Report, Symbolics, Inc., 21150 Califa Street, Woodland Hills, California, 1981.
5. Waterman, Donald A., A guide to Expert Systems, Reading, Massachussets: Addison Wesley Publishing Company.
6. Winston, Patrick H., Artificial Intelligence, (2nd ed.), Reading, Massachussets: Addison Wesley Publishing Company, 1984.

N89-15579

A CLIPS Prototype for Autonomous Power System Control

James M. Vezina Leon Sterling
Center for Automation & Intelligent Systems Research
Computer Engineering & Science Dept
Case Western Reserve University
Cleveland, Ohio 44106

August 1988

The model of the system assumes a constant power source and loads (experiments) whose power demands exceed the supply. Experiments are described by their: name, power consumption, time for a complete run, present status and the state of the load. The power consumption of each load is set at a constant level but can be dynamically modified by the operator. The status specifies if the experiment is running, paused, completed or failed. The state compensates for the lack of actual feedback sensor data, by signifying the stability of the load. Experiments are scheduled to keep as many running as possible with the current system limitations. A graphics oriented user interface is embedded into the rule-based system to enable an operator to easily experiment with the system.

1 INTRODUCTION

With limited resources and the high cost of electrical power on the space station, it is essential to have a highly reliable and robust system to manage the scheduling and restoration of the power system. An expert system, embedded with the tried and true terrestrial algorithmic decision aids, will be able to maintain the power system with little assistance from airborne or ground support. CLIPS (C Language Integrated Production System), developed by NASA, is used to demonstrate how a rule-based system could be used to achieve this lofty goal. The prototype simulates a simple model of the power system and adaptively schedules the various loads. Once a configuration is planned, the system monitors the loads for variation in power consumption, potential failure and

completion. Upon entering one of these states, the system is reconfigured to compensate for the change.

A team maintains a city's electrical power with various feedback sensors and mathematical algorithms. Conventional methods summarize power flow data for the entire network. Conventional techniques also provide avenues to investigate the affect of possible enhancements to the system. A power system could be maintained (at least in part) by capturing the team's expertise and integrating it with the appropriate traditional techniques. This will not replace the team, but will greatly aid them in recognizing and responding to problems that arise. People can be removed from the day-to-day problems of maintenance, thus enabling them to concentrate on improving the efficiency of the system.

This project demonstrates how a rule-based expert system, embedded with user defined functions (written in C), could autonomously control the space station power system. The actual model has been greatly simplified, but still can be used to explore various control strategies. C Language Integrated Production System (CLIPS)[2][3], developed by NASA, provides a low cost, yet highly portable, expert system shell to carry out our initial endeavors. The ease of integrating algorithmic routines in CLIPS, enables the system to access traditional tools and techniques. This simple, yet robust system provides a platform to explore new concepts in autonomous space station power system control.

The current model consists of a single power source, primary buss and various loads (although only four are displayed on the screen). The power consumption of each experiment is divided into four classes: **normal, warning, critical, failure**. **Warning** indicates the experiment is using more power than anticipated. If the experiment's state is unstable, then this level notifies the operator that a potential problem may exist. Once an unstable experiment enters a **critical** level, it is assumed that it will fail and is therefore aborted. To prevent harmful side-effects to the rest of the system, an aborted experiment will never be rescheduled. If a stable experiment reaches the **critical** level, the operator is notified that a *good* experiment is consuming an abnormally high amount of power. The actual model of the system will be discussed in more detail, along with an explanation of the expert system.

2 PROBLEM DESCRIPTION

The model uses a constant source of power to supply the demands of the experiments. The system enables the operator to vary the available power, but it does not automatically simulate the various fluctuations that would occur in an orbiting space station. The more detailed aspects of a power distribution system, such as a spike on a line or insufficient line load[1], were beyond the scope

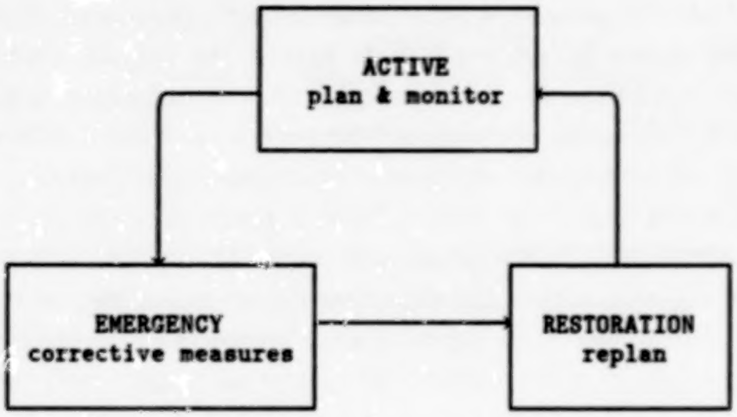


Figure 1: Power System Modes of Operation

of this work. The demonstration considers a primary power buss with auxiliary paths going to the loads. Later we will show how a user can modify the map that specifies the system configuration. Figure 1 shows the three basic modes of operation: **normal, emergency** and **restoration**. Normal mode consists of scheduling the operation of loads and monitoring the system. If a failure occurs (or is predicted to occur) the system enters emergency operation. Drastic measures must be taken to insure the integrity of the power system as a whole. An unexpected variation in the available power, or an unexpected event in the system could trigger an emergency situation. Restoration operations get the system back on track and operating normally. The system must decide which loads should be started, aborted, temporarily stopped or just left alone.

The primary goal of the expert system is to supply power to as many experiments (loads) as possible. This is not the best optimal goal for scheduling[4][5]. but was sufficient to demonstrate a type of scheduler embedded into the rule-based system. The system monitors each active load and notifies the operator if an experiment *moves* toward potential failure. A load can operate at one of five levels: **dormant, normal, warning, critical,** and **failure**. If the experiment is not running, due to a problem or scheduling constraint, it is considered dormant. The anticipated amount of power consumption is, of course, the normal level. The warning level is set to indicate an experiment is approaching failure. If the experiment consumes enough power to be *close* to failing, it becomes critical. Depending on how the system views the *correctness* of the experiment, this level would push the system into emergency mode. Failure is beyond the set limits of the breaker for the particular experiment.

In this model, the various levels are the same for all of the experiments (although this does not have to be the case). The switches (breakers) are considered to be intelligent and able to measure a potentially failing experiment. If the experiment is operating correctly, but happens to exceed the breaker capacity, it is assumed that this condition was an accident and will not cause any problems. The experiment will continue running (the switch is closed). However, if an intelligent sensor determines that the power consumption is unstable, it will terminate the experiment before the failure level is reached, avoiding possible damage to other experiments. This method now moves the system into restoration operations. During restoration, the experiments are scheduled to take advantage of any available power. The experiments themselves are considered to be too varied and specialized to be maintained by the on-board crew, therefore diagnosing and repairing a failed experiment is considered infeasible. With this assumption, the pertinent information for each experiment is its power requirements and time-to-run. The system is also robust enough to take advantage of interruptable experiments. During restoration, the experiments are scheduled to take advantage of any available power.

The scheduler dynamically runs as many experiments as possible. If power is available, the scheduler will iteratively look for a waiting experiment that can be supported under the current constraints and start it running. This continues until there is not enough power to support any of the remaining loads. If, for some reason, the running experiments exceed the power limits of the system, unsupportable experiments are put back into the wait state or if necessary, aborted. The waiting experiments would later continue from the point they were paused. When an experiment completes successfully or is aborted, it is stopped and will not be scheduled again. At each step, the power consumption values are simulated for the entire system.

The expert system monitors the space station power system simulation. If an operating experiment nears a potentially fatal level, the operator is warned of the approaching problem. The state of the experiment can be considered good or bad. An operator will be warned if a *good* experiment enters a critical state (nearing the maximum level of the breaker). If this experiment blows the breaker, it is assumed that the load will not affect any part of the system, and the breaker is closed. When a *bad* experiment enters the warning level, the operator is notified. Upon entering a critical state, the experiment is aborted to prevent possible degradation to the system. When an experiment successfully terminates or fails, the operator is notified and appropriate action is taken. The diagnostics in this system are straight-forward, *if it fails (or is predicted to fail) then it is aborted.* This approach is disastrous in most applications, but here it is quite reasonable. Groups from all over the world will have their experiments running on the space station. The crew in the space station will not have the time or expertise to be capable of repairing failed equipment in an experiment. The availability of parts will also limit an operator's ability to repair the transient experiments.

The user-interface (as shown in figure 2) displays the current state of the system. It was intented to display as much information as possible in an easy to understand format. In an actual application, it would not be practical to continuously display all of the data, but this project was designed as an learning tool or experimental platform. At the start of each time interval, the operator can elect to modify any of the parameters of the experiments. In this way, the user can induce potential problems into the power system and experiment with the corrective actions taken by the expert system.

As can be seen in figure 2, there is a **dynamic help bar** containing the commands or responses that are possible at a given time (the commands themselves prompt the user for all input). When modifying (editing) an experiment, the dynamic help bar displays all of the possible commands and alters the prompt accordingly. If the operator elects to increase the power consumption of an experiment, the system prompts for the new value, while the help bar indicates the appropriate response (an integer value) and range. The operator can also access a brief on-line tutorial describing the entire system.

The user-interface is primarily written in C, using the curses library. These commands were embedded into the CLIPS shell and are called via rules. The changes made by the user are asserted directly into the knowledge base of the shell. Rules exist to take these changes and appropriately *edit* the simulation database.

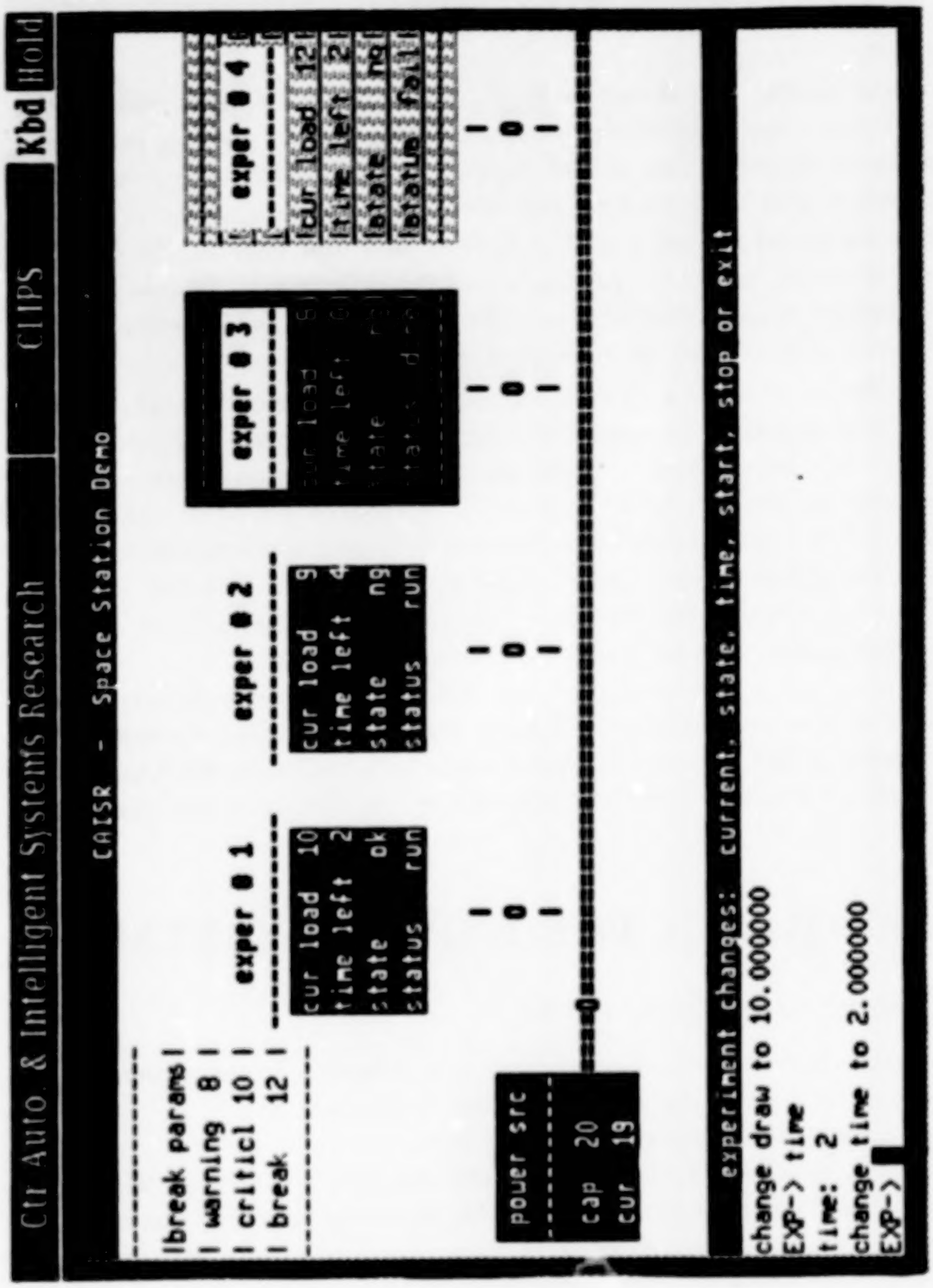
3 DETAILS ON THE EXPERT SYSTEM

3.1 Facts - Data Structures

The data structures are broken down into four groups: switches, experiments, power supply and general facts. The facts are in the form of:
(type-of-node node-name slot-name value).
The type-of-node indicates what type of node the fact is concerned with (e.g. experiment), and the node-name specifies the particular node (e.g. name of the experiment). The slot-name determines what the fact is about and its value. An example of a slot-name would be *status* with the value of *wait* or
(experiment exper_1b status wait).

The switches have a fact for their electrical current and another for their state (i.e. open, closed or blown). Each experiment has 5 slots of information (facts): current load, time, time-to-end, status and state. Figure 3 contains an example of experiment 2 and its corresponding switch. The time slot contains

ORIGINAL PAGE IS OF POOR QUALITY



```
experiment changes:  current, state, time, start, stop or exit
change draw to 10.000000
EXP-> time
time:  2
change time to 2.000000
EXP->
```

Figure 2: An example session

```
(switch  s1  state    open)
(switch  s1  current  0)

(experiment  exper_2  draws  10)
(experiment  exper_2  time   5)
(experiment  exper_2  time_to_end  unknown)
(experiment  exper_2  state  ok)
(experiment  exper_2  status wait)

(connect s1  exper_2)
```

Figure 3: An example of the facts describing an experiment and its switch

the total amount of time till the experiment is finished. In the figure, the time-to-end has a value of **unknown** to indicate that is has no scheduled end time (because it is not running). Status of the experiment can be: wait, run, done or failed. The state of the experiment is used to simplify the simulation of the power system. As mentioned earlier, a switch can measure the reliability of the experiment. Instead of simulating this operation, each experiment is listed as: OK or NG (No Good for not operating as expected). This is all the information necessary schedule and maintain the experiments.

The configuration of the system is also given by the facts. There is a single fact for each connection in the power system. Figure 3 shows the connection between the switch and the experiment. By altering these simple facts, the power system network can easily be redesigned.

3.2 Rules

The rules of the system control the simulation, the user interface and any algorithmic functions embedded in the system. The knowledge base simulates the power system model to enable the user to easily reconfigure the network by altering a few facts. A traditional algorithmic technique could prove to be more accurate and much faster, but would lose the flexibility and ease of the current method. The system traverses the network calculating the power level at each node. The user interface was embedded directly into CLIPS and can be viewed as a group of new shell functions. CLIPS greatly simplifies the integration of the traditional methods with the expert system. Data can easily be passed to and from the user defined function by simple *get* and *assert* type commands.

The rule format is shown in the following figure (figure 4). If the current passing through a switch is above the indicated failure level, then the switch must not be operating correctly, otherwise it would have blown. The semi-colon (";") delineates comments on the rule. The rule name is **Switch_bad** and is used if the breaker appears to be fused closed. The first six lines after the rule name are the antecedents of the rule. These must all be satisfied in order for the rule to be fired. The question mark ("?") indicates that the following is a variable and should be set to the appropriate value of a matched fact (CLIPS uses the data driven Rete matching algorithm). The ampersand-colon indicates that the match must also satisfy the additional constraints that appear in the parentheses. The three lines after the arrow ("*arrow*") are the consequences, executed when the rule is fired. In this example, the consequences assert three new facts into the knowledge base. The first and second will be sent to the operator (via the error handler and then the user-interface). The last line will fire another rule to halt (abort) the experiment. Although this example does not have a specified priority, any rule can be given a priority by adding: **(declare (salience 100))**. This line declares that the priority of the rule is 100 and will internally adjust the order for inferencing the rules.

The rules for monitoring and restoration can be broken down into the seven basic rules listed in table 1. The table has the rules indexed in each row of the table. The first three columns show the more pertinent information for deciding if an experiment "looks" as if it will present any danger to the system. SWITCH CURRENT gives the symbolic value for the level of power the experiment is consuming. SWITCH STATE represents the expected state of the network and the experiment (should the experiment be connected to the network, i.e. switch *closed*). The EXPERIMENT STATE, as mentioned earlier, indicates the feedback information on the power flow to the experiment. This is either good or bad (OK or NG). The last two columns are the consequences to the scenarios. The REACTION represents the expert system response to the operator or the power system. The user could be notified or appropriate action might immediately be taken on the network itself (e.g. disconnect the experiment from the rest of the system). The NEW EXP. STATE indicates the new status of the experiment, **fail** (if the experiment is aborted) or **same** (if it is not altered). Emergency handling rules and a variety of system maintenance rules complete the rule-based controller, and the discussion on this project.

4 ACKNOWLEDGEMENTS

This work was supported in part by a grant from the NASA Lewis Research Center (grant number: NAG 3-787). I appreciate all of the help I received from NASA Lewis, especially Les Burke and Jim Kish, and the continued help and

```
;
;**********************************************
; RULE:  6
;  Switch:    FAILURE
;             closed
; Experiment: NG
; DO:        NOTIFY / OPEN_SWITCH
;
(defrule Switch_bad
   (switch ?switch_name current ?i_value)
   (switch current failure
        ?failure_value&:(>= ?i_value
                            ?failure_value))
   (switch ?switch_name ?experiment_name)
   (experiment ?experiment_name status run)
   (experiment ?experiment_name state ng)
=>
   (assert (info experiment ?experiment_name
                            halted))
   (assert (info switch ?switch_name fused))
   (assert (stop_experiment ?experiment_name
                            fail)))
```

Figure 4: An example of a rule in CLIPS

Diagnostic & Restoration Rule Table					
Rule number	Switch current	Switch state	Experiment state	Reaction	New Exp. state
1	ok	closed	ok	——	same
2	warn	closed	ng	notify	same
3	critical	closed	ok	notify	same
4	critical	closed	ng	notify open-switch	fail
5	failure	closed	ok	notify	same
6	failure	closed	ng	notify open-switch	fail
7	——	blown or open	(running)	notify	fail

support from Eric Bobinsky. I would like to thank the Cleveland Advanced Manufacturing Program for their aid in supporting this project. Finally, I would like to acknowledge the computer resources provided by the Center for Automation and Intelligent Systems Research at Case Western Reserve University.

5 REFERENCES

[1] DyLiacco, TE *The Adaptive Reliability Control System* IEEE PAS-104(12) pp 3423-2427, Dec. 1985

[2] Giarratano, Joseph C. **Clips User's Guide** Sept 13, 1987

[3] Giarratano, Joseph C. **Clips Reference Manual** Sept 13, 1987

[4] Touchton, Robert A *Common Module Dynamic Payload Scheduler Expert System* **Proceedings of the Twenty-first Intersociety Energy Conversion Engineering Conference**, August 1986, vol III pp 1785-1790

[5] Washington, Sylvia **Optimal Load Management for the Space Station Power System** Masters Thesis, Case Western Reserve University, 1987

N89-15580

'A HARDWARE IMPLEMENTATION OF A RELAXATION ALGORITHM TO SEGMENT IMAGES'

by

ANTONIO G. LODA' , HEGGERE S. RANGANATH
2003 Fulton Dr., Huntsville , Computer Science Department
University of Alabama in
Huntsville

ABSTRACT

Relaxation labelling is a mathematical technique frequently applied in image processing algorithms. In particular, it is extensively used for the purpose of segmenting images. The paper presents a hardware implementation of a segmentation algorithm, for images that consist of two regions, based on relaxation labelling. The algorithm determines, for each pixel, the probability that it should be labelled as belonging to a particular region, for all regions in the image. The label probabilities ('labellings') of every pixel are iteratively updated, based on those of the pixel's neighbors, until they converge. The pixel is then assigned to the region correspondent to the maximum label probability. The system consists of a control unit and of a pipeline of segmentation stages. Each segmentation stage emulates in the hardware an iteration of the relaxation algorithm. The design of the segmentation stage is based on commercially available digital signal processing integrated circuits. Multiple iterations are accomplished by stringing stages together or by looping the output of a stage, or string of stages, to its input. The system interfaces with a generic host computer. Given the modularity of the architecture, performance can be enhanced by merely adding segmentation stages. The processing speed is near real time.

I. Introduction.

Image analysis is concerned with the description of images and the recognition of objects. Traditionally, image analysis has been applied extensively in the space exploration field, for example in the analysis of pictures taken from spacecrafts or satellites. The first and foremost step in an image analysis algorith is segmentation. Segmentation consists of processing an image into meaningful regions. Therefore the success of image analysis depends largely on the accuracy of the segmentation algorithm. A widely accepted segmentation technique is called 'relaxation labelling'. A full description of this algorithm is deferred to section II. Among its virtues, relaxation lends itself very well to hardware implementation.

This paper describes the architecture of a blob detector, a system that segments images characterized by two regions. The system is a hardware implementation of the relaxation labelling algorithm , in its classical probabilistic form. In section II, the relaxation algorithm is described in detail. In section III, the architecture of the system is defined. The conclusions are drawn in section IV.

II . Classical probabilistic relaxation labelling.

The following discussion is to familiarize the reader with a segmentation algorithm called 'classic probabilistic relaxation labelling' [1]. This segmentation technique is here described in a step by step fashion. The discussion sets the background for the definition of the hardware implementation of the algorithm, in section III.

II.1. The algorithm.

Let $D=\{px(i,j),i=1,..,n1,j=1,..,n2\}$ be a digital image consisting of m regions $c_1,..,c_m$. Let g_k be the average gray level for region k. What follows is a step by step procedure to segment the image:

STEP 1: For every pixel px(i,j) a set of probabilities $P^{(0)}(i,j,1),..,P^{(0)}(i,j,m)$, or 'probability function' $P^{(0)}$, is computed as follows:

$$P^{(0)}(i,j,k)=(1/\mathrm{abs}(px(i,j)-g_k)+\varepsilon)/\sum_{k=1}^{m}(1/\mathrm{abs}(px(i,j)-g_k)+\varepsilon) \qquad k=1,..m \quad [2.1.1]$$

where $P^{(0)}(i,j,k)$ represents the probability that pixel px(i,j) belongs to region c_k, ε is non zero constant and

$$\sum_{k=1}^{m} P^{(0)}(i,j,k)=1 \qquad \{2.1.2\}$$

The following steps are iterated. The formulas are generalized for iteration r.

STEP 2 : For every pixel px(i,j), a set of 'compatibility coefficients' $c^{(r)}(i,j,k,i1,j1,k1)$, or 'compatibility function' $c^{(r)}$, is defined, where $i1=1,..,n1$, $j1=1,..,n2$, $k,k1=1,..,m$, and $(i1,j1)=(i,j)$. $c^{(r)}(i,j,k,i1,j1,k1)$ represents the compatibility between the assignment of px(i,j) to region c_k and that of p(i1,j1) to c_{k1}. The compatibility function is basically a heuristic evaluation of the validity of a pixel's region assignment (labelling), on the basis of the labellings of the rest of the image pixels.

Two commonly used assumptions in the formulation of the compatibility function are as follows:

1. Only the neighborhood pixels are relevant to the classification of the pixel under scrutiny. Therefore

$$c^{(r)}(i,j,l,i1,j1,k1) \begin{cases} = 0 & \text{if } i-1 < i1 < i+1 \\ & \text{and } j-1 < j1 < j+1 \\ = 0 & \text{otherwise} \end{cases} \qquad \{2.1.3\}$$

2. The compatibility function is 'space invariant' that is, for every integer ii, jj such that px(i+ii,j+jj) and px(i1+ii,j1+jj) belong to D,

$$\begin{aligned} &c^{(r)}(i,j,k,i1,j1,k1) = c^{(r)}(i+ii,j+jj,k,i1+ii,j1+jj,k1) \\ &\qquad i=1,..,n1;\ j=1,..,n2; \\ &\qquad i-1 < i1 < i+1, \text{ and } j-1 < j1 < j+1 \end{aligned} \qquad \{2.1.4\}$$

Several definitions have been introduced for the compatibility function, one is given below. Let

$$C^{(r)}(k,ii,jj,k1) = c^{(r)}(i,j,k,i+ii,j+jj,k1)$$
$$i=1,..,n1,\ j=1,..,n2,\ -1 < ii < 1,\ -1 < jj < 1 \qquad (2.1.5)$$

Let $p1^{(r)}(k)$ represents the a priori probability of an image pixel belonging to region c_k. Let, also, $jp^{(r)}(k,ii,jj,k1)$ be the joint probability that an image pixel belongs to region c_k and its neighbor, at the orientation specified by (ii,jj), belongs to region c_{k1}.

We define

$$C^{(r)}(k,ii,jj,k1)=[\log R^{(r)}(k,ii,jj,k1)], \qquad (2.1.6)$$

where

$$R^{(r)}(k,ii,jj,k1)= [jp^{(r)}(k,ii,jj,k1)/(p1^{(r)}(k)*p1^{(r)}(k1))] \qquad (2.1.7)$$

For practical purposes, the values of the compatibility function are truncated to the interval [-1,1].

STEP 3: A set of supporting coefficients $Q^{(r)}(i,j,1),..,Q^{(r)}(i,j,m)$, or 'support function' $Q^{(r)}$, is computed as follows:

$$Q^{(r)}(i,j,k)= (1/8) \sum_{i1=i-1}^{i+1} \sum_{j1=j-1}^{j+1} \sum_{k1=1}^{m} C^{(r)}(k,i1-i,j1-j,k1)P^{(r)}(i1,j1,k1) \qquad (2.1.8)$$

$Q^{(r)}$ represents the contribution of the total relevant environment of pixel px(i,j) to $P^{(r)}(i,j,k)$.

STEP 4: $P^{(r)}(i,j,k)$ is updated as follows:

$$P^{(r+1)}(i,j,k) = [P^{(r)}(i,j,k)[1+Q^{(r)}(i,j,k)+\varepsilon]] / \sum_{k=1}^{m} P^{(r)}(i,j,k)[1+Q^{(r)}(i,j,k)+\varepsilon] \qquad \{2.1.9\}$$

Each pixel px(i,j) is then assigned to the region $c_{K^{(r)}}$, where $K^{(r)}$ is such that the probability $P^{(r)}(i,j,k)$ is maximum for $k = K^{(r)}$.

The iteration is repeated until the labellings converge. Alternatively, one can stop the algorithm after a fixed number of iterations has been executed, or according to some other termination scheme.

III. The blob detector.

The algorithm presented in the previous section is the centerpiece in the design of the blob detector. The system is intended for scientific and industrial applications. The speed required in these applications cannot be normally accomplished by general purpose computers. Computers based on special purpose architectures, such as array processors or systolic arrays, are better suited but also result in costs often not justifiable in the context of simple applications.

The system defined in this paper derives its speed from its dedicated architecture. By optimizing the design for a specific algorithm the system complexity is reduced as well. Also, the architecture is pipelined, since the promptness of the result is not as important as the system's throughput. Finally, the design achieves expandability through modularity and is intended as a peripheral to a commercial general purpose personal computer, to facilitate its use.

As a result of these design choices the blob detector is a high speed, low-cost, low-complexity system, configured as a peripheral to a personal computer.

III.1. General system architecture

The system consists of a microcontroller (MC) and of the relaxation engine (RE) (Figure 1). The microcontroller controls the synchronization of all system operations, through the system control (SCB) and i/o (SIOB) buses. It communicates with the host computer through the host interface(HI). The latter allows the host to upload the image data, request the execution of the segmentation procedure and download the processed image. The segmentation engine is responsible for the execution of the segmentation algorithm. The input and the segmented images are received and transmitted over the system i/o bus .

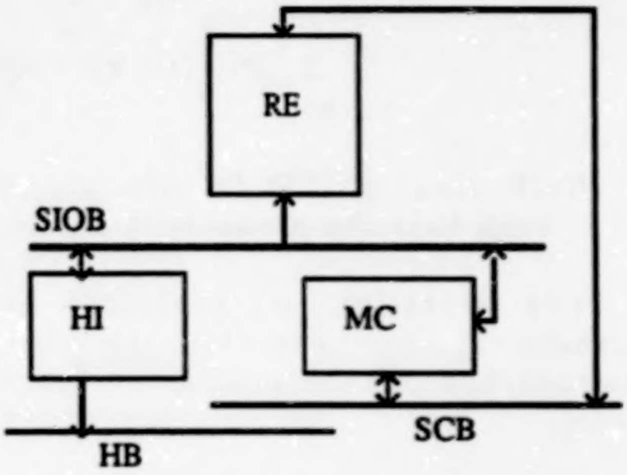


Figure 1. The system architecture.

The function of the microcontroller is to coordinate the overall system operation and the communication with the host computer. Furthermore, it is designed to have enough processing power to perform some post-segmentation simple image processing tasks, should this be required.

The architecture of the microcontroller is based on a commercial 32-bit microprocessor (MP), coupled with a math coprocessor (MCP) (Figure 2). A memory management unit (MMU) oversees the microcontroller's accesses to the memory system (MS). The memory system consists of both RAM and ROM type memories. The ROM memory is necessary for system initialization and for storing the system algorithms. The RAM memory is used for storing the segmented image and user specific algorithms dowloaded from the host

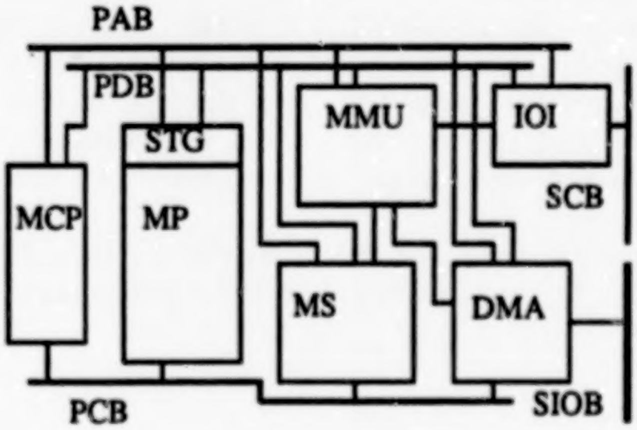


Figure 2. The microcontroller.

system. The timing of the microcontroller as well as of the entire system is generated by the system timing generator (STG).

The host interfce consists solely of the circuitry necessary to insure the electrical continuity between the host interface and the system and to handle the handshake protocols.

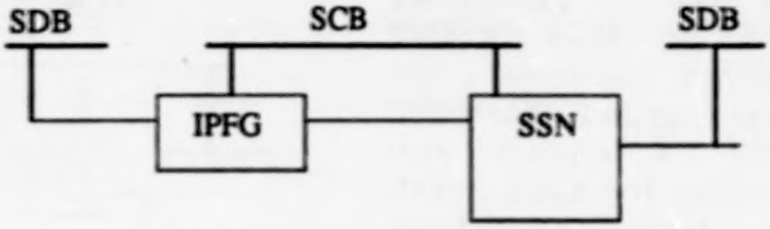


Figure 3. The relaxation engine.

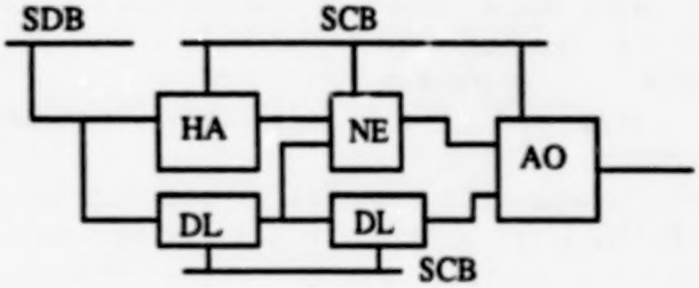


Figure 4. The initial probability function generator.

III.2. The relaxation engine

The algorithm described in section I is executed by the relaxation engine (Figure 3). The image data, arriving from the host system via the microcontroller, is initially analized to derive the initial probability function. This task is performed by the initial probability function generator (IPFG) (Figure 4). This circuitry determines the function $p^{(r)}$ as defined in [2.1.1]. The image data is first processed by the histogram analizer (HA), which determines the average gray level values for the two regions that are to be segmented in the picture.

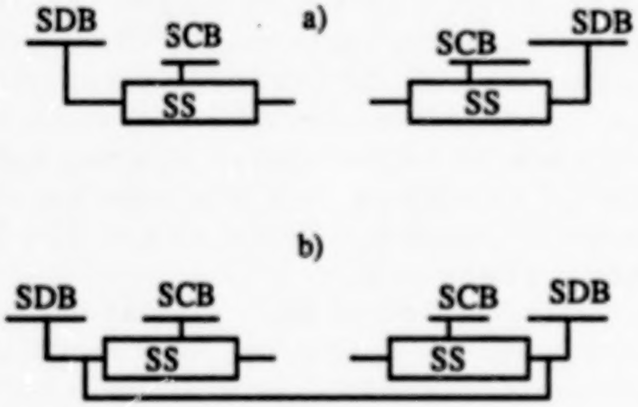


Figure 5. Segmentation stage networks:
a) chain configuration b) loop configuration.

The image data, delayed by the histogram analyzer processing time is then passed through the neighborhood extractor (NE), a circuit that latches, for each pixel, its eight neighbors. These pixel values, together with the region average gray level values, are then fed through a series of arithmetic operators (AO) arranged to implement {2.1.1}. Since the image only consists of two regions and therefore $P^{(r)}(i,j,0) = 1 - P^{(r)}(i,j,1)$, only $P^{(r)}(i,j,1)$ is computed, for every pixel. It is possible to execute to execute these operations in real time because components are now available that execute multiplications and divisions at the rate of one every 40ns. The neighborhood extractor (Figure 3) is based on a series of delay lines (DL), which are circuits that output at every instant the data received in input n clocks earlier, where n is the length of the line. Each delay line is implemented using shift registers. Delay lines are also used in the design to synchronize the pipeline.

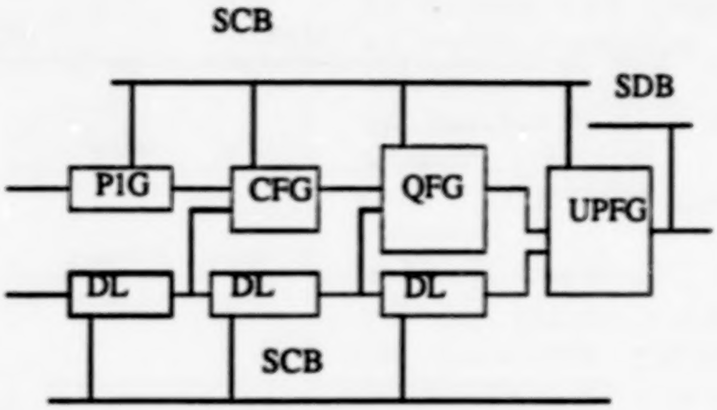


Figure 6. The segmentation stage.

The second functional block of the relaxation engine is the segmentation stage network (SSN). This subsystem iterates the probability updating scheme described in section I. It consists of a network of segmentation stages , each capable of executing one iteration of the relaxation algorithm. These stages can be modularly connected in a variety of ways. Two configurations are displayed in Figure 5. The first (Figure 5.a.) consists of four units cascaded, so that the output of a stage is the input of the next one. The second configuration (Figure 5.b.) sacrifices some of the system throughput

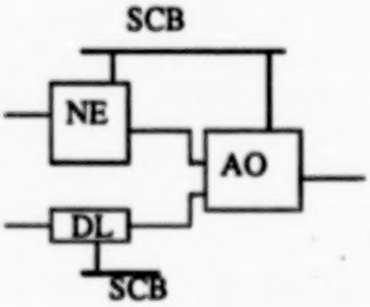


Figure 7. The compatibility function generator.

of the by iterating the algorithm by looping the output of a string of two stages to its input, as many times as is desired.

Each segmentation stage (Figure 6) is responsible for executing one iteration of the relaxation algorithm, that is, for obtaining $P^{(r)}(i,j,k)$ from $P^{(r-1)}(i,j,k)$. The first task of the updating unit is to determine the a priori probabilities p1(0) and p1(1) of the two image regions. The probability function and the a priori probabilities are then output to the compatibility coefficient generator (CFG). This circuitry determines the image compatibility function by running each pixel neighborhood through a network of arithmetic operators, arranged in a sequence, such to reproduce the calculations defined in {2.1.6} and {2.1.7} (Figure 7). Once the compatibility function is computed, it is output, together with the probability function, to the support function generator (QFG). The support function and the probability function, finally are processed by the updated probability function generator (UPFG) to produce the updated probability function. Both the support and the updated function generators are networks of arithmetic operators arranged so to perform {2.1.8} and {2.1.9}.

IV. Conclusions.

In the previous pages we presented the architecture of a blob-detecting system. The system, based on a pipelined processing scheme, allows for real time segmentation of 'blobby' images for scientific and industrial applications. The system is designed to be an inexpensive image analysis peripheral to a commercial personal computer. The design can be expanded, with little effort, to add the capacity to execute other image processing algorithms, characterized by the application of the same procedure on all pixels, and that operate on a neighborhood basis. Algorithms such as template matching, for the recognition of objects, fall in this cathegory. This expansion can be achieved by replacing the compatibility function circuitry with memory, which can be loaded with the desired operator, and replacing the dedicated arithmetic operators network with a programmable digital signal processor.

V. Literature.

[1] Kittler J. and Illingworth J., "Relaxation labelling algorithms - a review," *Image and vision computing*, Vol.3 No.4 (1985), pp. 206-216.

N89 - 15581

Using AGNESS (A Generalized Network-based Expert System Shell) for matching images

Ting-Chuen Pong, Chung-Mong Lee and James Slagle

Department of Computer Science
University of Minnesota, Minneapolis, MN 55455

The image correspondence problem has generally been considered the most difficult step in both stereo and motion analysis. Stereo vision is useful in determining the three-dimensional positions of points on visible surface in a scene. Motion analysis is useful in determining the spatial and temporal relationships of objects in an environment. Besides stereo and motion analysis, image correspondence problem. Most of this work is based on point or local area properties of the observed gray level values in two-dimensional images.

In this paper we describe a global and general approach to this problem by using a knowledge-based system. The knowledge it uses consists of both physical properties and spatial relationships of the edges and regions extracted from the given images. The physical component depends on features of the edge or region) in isolation. The spatial component involves the set of edges and regions adjacent to a given edge (or region) of the first image and the set of edges and regions adjacent to each potentially matching edge (or region) of the second image; thus the spatial context of each edge or region is considered. A computation network is used to represent this knowledge, it allows the computation of the likelihood of matching two edges or regions with logical and heuristic operators.

An expert system shell called AGNESS (A Generalized Network-based Expert System Shell) is used to build a prototype system. All the extracted feature values (for example length, orientation, contrast, etc.) are automatically input to a computation network at the beginning of the process. The comparison between the feature values will then take place in the intermediate nodes and the final result of how well each edge or region matches its potential edges is calculated at the top node. The entire control process of the system consists of two phases. The first phase determines how similar each edge or region) is to each of its potential edges (or regions). the second phase will then complete the global evaluation procedure, that involves conflict resolution.

PRECEDING PAGE BLANK NOT FILMED

N89 - 15582

AUTOMATIC INSPECTION OF ANALOG AND DIGITAL METERS IN A ROBOT VISION SYSTEM*

Mohan M. Trivedi
Suresh Marapane
ChuXin Chen

Electrical and Computer Engineering Department
The University of Tennessee
Knoxville, TN 37996-2100

ABSTRACT

A critical limitation of most of the robots utilized in industrial environments arises due to their inability of utilize sensory feedback. This forces robot operation in a totally preprogrammed or teleoperation modes. In order to endow the new generation of robots with higher levels of autonomy techniques for sensing of their work environments and for accurate and efficient analysis of the sensory data must be developed. In this paper detailed development of vision system modules for inspecting various types of meters, both analog and digital, encountered in a robotic inspection and manipulation tasks are described. These modules are tested using industrial robot having multisensory input capability.

1. Introduction

Advanced robotic systems capable of performing a variety of tasks in complex unstructured environments will have to possess sophisticated sensory capability for acquiring information about their work environments. Also required is the associated capability for analyzing such information in an accurate and efficient manner. Robotic system with sophisticated sensory capability will be of particular utility in tasks such as automated assembly [5], inspection and manipulation in hazardous environments such as nuclear plants, [7] or space based platforms. Types of sensors which can be utilized in such systems include vision, range, proximity, tactile, force and torque, etc. [3]. Of these, vision sensory modality is recognized as the one providing rich characterization of work environment with various types of relatively inexpensive and well engineered camera systems.

The specific area of research reported in this paper deals with a vision system that has been designed and developed to perform various inspection and manipulation tasks associated with a control panel. The panel contains a number of displays, both analog and digital, and controls like switches and valves. In addition to the above panel the test bed for our research includes an industrial robot capable of sensing the environment with vision, range, proximity, force and torque and touch sensors. The main tasks performed by the vision system include: automatic location of the test panel, positioning of the robot arm to acquire appropriate input images of the panel, automatic recognition of all objects

---

*This research is supported by the Advanced Technology Development Division of the U. S. Department of Energy under grant No. DOE DE-FG02-86-NE31968.

PRECEDING PAGE BLANK NOT FILMED

appearing on the panel and determination of their 3-dimensional location. Once the objects are located the system has to acquire finer resolution images of individual objects to determine their status. In this paper we shall present the detailed procedure developed to "read" various types of analog and digital meters. The paper includes results of several experiments carried out to verify the robustness of the system in performing automatic inspection and manipulation tasks. It is believed that the robot vision system can be utilized to perform tasks in a number of application domains including space.

2. Vision Guided Robotic Inspection and Manipulation

The main goal of a robot vision system is to provide an accurate interpretation of a scene utilizing images of the scene as the primary source of input. The interpretation can be provided in a variety of forms and at different levels of abstraction. A useful form of interpretation may include an object location map where different types of physical objects appearing in the scene are independently recognized and accurate locations of these objects in the scene are determined. Also, of utility is the information regarding the status or condition of an object. Design of a computer vision system that can perform such object recognition and scene interpretation is a complex and challenging task. The main difficulty underlying the task comes from the fact that images are 2-dimensional projections of the 3-dimensional real scene and innumerable combinations affecting the illumination source, scene and sensor parameters can result in the same observable value of recorded image intensity.

In order to make the above problem computationally tractable model-based approach to computer vision is proposed [6]. The approach requires knowledge of a series of models associated with objects which are believed to appear in the scene. These models can be recorded in the knowledge-base of the system. Various features from the input images are extracted by using low- level, general purpose operators. These operators should be robust in extracting image features containing meaningful information about the objects. Finally, a correspondence is sought between the image derived features and scene domain models to recognize the objects. This is accomplished by utilizing various decision making schemes in the matching module.

Development of autonomous systems for a variety of applications in an industrial environment will require major research efforts to resolve many complex issues. We have undertaken an approach, which we believe allows making incremental progress towards the eventual development of such a system. Our initial research effort is directed towards researching issues associated with acquisition and analysis of multiple sensory data using a robotic system. This is accomplished by focusing on the development of an autonomous system that is capable of performing various inspection and manipulation tasks associated with a typical panel. For example these tasks can range from reading of various meters and displays, operating different types of switches and controls. Also, included are tasks associated with valve operation. Teleoperation or automatic operation of valves in nuclear power plants is recognized as one of the important desired capabilities of a robotic systems. Design of the panel was done in consultation with a team from a nuclear power plant developer, using all "off-the-shelf" components. Our experimental set-up includes a test panel, a robot having multiple sensory capability, computers, and various manipulation tools. The test panel and the robot with various sensors mounted on the arm are shown in Figure 1.

Typical autonomous robot operation will involve the following. The robot first identifies the exact geometrical position of the panel using a camera calibration program. Next it uses a computer vision system to develop an object location layout map for various devices appearing in the panel. The task command for the robot is provided through the binary coded lights of an LCD display. After decoding the command the robot performs requested inspection or manipulation task.

Robustness and ease in expandability to accomodate changes in the task environment are two key features guiding the development of the vision system. The system is compartmentalized in two basic groups of procedures. The first group consists of general purpose procedures for camera calibration, image acquisition, knowledge acquisition, image segmentation, matching, and robot arm movements. The second group consists of special purpose procedures mainly designed for determining status of individual objects.

3. Recognition of Object Status

Depending upon the type and nature of the object the camera mounted on the arm is moved to take images using orthogonal viewing geometry. These images are analyzed to determine the status of the object. The objects appearing on the test panel and the type of status information associated with each one of them are listed in Table 1.

Table 1. List of objects appearing on the test panel and their status information.

OBJECT TYPE	STATUS
1. Light	On/Off
2. Analog Meter	Needle Reading
3. Digital Meter	7-Segment Code
4. Valve	Position of the Holes
5. Slider Control	Position of the Sliding Arm
6. Push Button Switch	On/Off
7. Toggle Switch	On/Off

In order to account for changes in the task environment one will require an additional knowledge acquisition session to update the knowledge base and incorporation of the appropriate routines to determine the status information of the objects added to the knowledge base.

Status recognition of three object types are considered in this paper. They are two types of analog meters and digital (LCD) meter. Once the objects are identified by the object recognition module further processing is required to recognize their status. The primary requirements of the object status recognizers are their robustness, accuracy, and speed. Incorporation of limited apriori knowledge about the objects greatly facilitate in meeting these requirements.

3.1 Reading an analog meter of type I

The main task in reading an analog meter is the determination of its needle orientation with respect to the horizontal direction. Once this angle is determined, the knowledge of the total swing angle of the needle for Full Scale Deflection(FSD) and the range of the meter enables one to compute the meter reading. It is assumed that the meter scale is symmetrical about the the vertical and that it is linear with respect to the angle. This assumption holds for many types of commercially available analog meters.

In this section it is assumed that the analog meter has been isolated from the rest of the image by the segmentor, and all further processing is performed within a window containing only the analog meter. The approach consists of two major steps.

The first step is the extraction of edges. For this, Marr edge detector [1] is chosen since it generates one pixel wide, closed contours. Since one can assume without loss of generality that the needle of a meter is a linear feature, the next step involves the analysis of all linear features within the processing window. For this, we choose Hough transforms [2]. The advantage of using Hough transforms includes it's relative insensitivity to noise and to gaps in the image of a line. This makes the procedure less sensitive to the results of the edge detector. In the slope-intercept representation of a line, however, the Hough parameter space becomes unbounded due to the slope and intercept becoming infinite for vertical lines. Since a bounded parameter space is desired for the analysis, a normal representation of a line is often used. In this representation the parameters are ρ and θ and

$$x \cos\theta + y \sin\theta = \rho \tag{1}$$

The parameter θ can be considered to be bounded between -θ_1 and θ_2. These angles are not required to be known exactly but the range needs to accomodate the total swing (i.e. all possible orientations) of the needle. Thus,it is reasonable to assume that this range of the parameter θ is known for a particular analog meter. Also, parameter ρ is bounded by the length of the meter. This implies that the parameter ρ is bounded in the range 0 to length of the processing window,w, known from the object recognition module. Therefore the analysis of linear features only requires to be performed within the bounded parameter space -θ_1 to θ_2 and 0 to w.

In the step 2, following edge detection, the linear feature analysis is performed within the bounded space -θ_1 to θ_2 and 0 to w. An implication of this quantization range of θ is that the horizontal linearities ($\theta = 90^\circ$) are not contained within the parameter space and therefore the horizontal edges are eliminated from further consideration. The elimination of false vertical linearities, linearities that does not correspond to the needle, is more involved since the needle itself may be vertically oriented. The key to the elimination of these false linearities is the observation that the vertical features that do not correspond to the needle have a ρ value which is either closer to 0 or w. Thus, seeking a local maxima, (θ_n, ρ_n), in the parameter space away from $\rho = 0$, say $\rho = 0 + \Delta_\rho$, and away from $\rho = w$, say $\rho = w - \Delta_\rho$, where Δ_ρ is a small number, will guarantee that the maxima is indeed due to the linearity of the needle.

Figure 2 shows a sequence of processing steps for an image where the analog meter occupies less than 2% of the total area of the image. The robustness of the procedure is clearly demonstrated in this experiment.

The accuracy of the final result is directly dependent on the accuracy of the angle of orientation of the needle. Hence, the accuracy of the result is dependent on the resolution of the parameter θ in the parameter space. In the experiments shown in this section the parameter θ was quantized to an 1 degree resolution. Since the analog meter had a resolution of 0.2 volts/degree this quantization results in a resolution of 0.2 Volts in the meter reading. This accuracy of the reading can be increased by finer division of the parameter θ at the cost of increasing execution time.

3.2 Reading an analog meter of type II

In this type of meters the needle is not pivoted at one end, but the needle moves across a horizontal scale. Thus, reading the meter requires determination of the needle position with respect to the left edge of the scale. Knowing the total length of the scale, L_s, the range of the meter, x_0 to x_f, and the needle position, l_s, the meter reading, x_r can be computed as:

$$x_r = \frac{x_f - x_0}{L_s} l_s + x_0 \tag{2}$$

In practise however, the lengths L_s and l_s can not be derived from the images since the segmentor can not isolate the inner scale from the rest of the meter. This requires reformulation of equation (2) using the derivable distances w, length of the meter(window), and l_w, the needle position with respect to the left edge of the meter(window). Using these distances x_r can be found as,

$$x_r = \frac{x_f - x_0}{w} l_w + x_0 - error \tag{3}$$

This error term can be computed using an image reading a known value, say x_0. In this case ,equation(2) will yield

$$error = \frac{x_f - x_0}{w} l_0 \tag{4}$$

Since the needle is a linear vertical feature, the analysis of linear features applies to this task as well. The detection of the needle requires only a minor change in the linear feature analysis step of the section 3.1. Now the Hough parameter θ is bounded between $0 - \Delta_\theta$ and $0 + \Delta_\theta$, where Δ_θ is a small angle. Notice that the $\theta = 0°$ corresponds to a vertical line. The bounds of the parameter ρ remains the same as in section 3.1, i.e., between 0 and w. This range of θ eliminates the horizontal features and the false vertical features are eliminated using the same rule of section 3.1, i.e., the local maxima in parameter space is required to be away from $\rho = 0$ or $\rho = w$.

Under these constraints, the local maxima ,(θ_n, ρ_n), will indeed correspond to the needle and $l_w = \rho_n$. Now we can compute the meter reading x_r using equation(3), if the error term has been previously computed as explained by equation(4).

The sequence of steps involved in processing an 128X128 image is illustrated in Figure 3. The original image is shown in Figure 3(a) and Figure 3(c) shows the line corresponding to the maxima in Hough space superimposed on the edge map (Figure 3(b)). These results suggests that the presented procedure is an effective method for reading this type of analog meters, provided that the algorithm can be trained for computing the error term in equation (3) using an image of known meter reading. The accuracy of the result is now dependent on the quantization of the parameter ρ.

3.3 Reading a digital (LCD) meter

This task actually consists of two recognition tasks. First, the digits needs to be identified, and secondly the decimal number represented by the individual digits needs to be identified. For recognition of digits we choose to use Fourier descriptors [4] for its size and rotational invariance properties. Once the individual digits are identified the decimal number is formed using an *ad-hoc* procedure explained in the next section.

As in the previous sections we assume that the processing is performed within a window containing the digital meter. Since the digits are best discriminated using their edge properties, the first step in processing is the extraction of edges. In order to represent the structural shape of the objects in a suitable form for the Fourier descriptors, this step is followed by a thinning process. In this step we use a modification of an algorithm developed by Zhang and Suen [8], for skeletonizing the edges. This algorithm is known to have some inherent drawbacks. One of the problems was due to the total elimination of small regions during the skeletonizing process. This limitation can however be overcome by simple modification.

In order to use the Fourier descriptors to identify the digits we require the digit to appear as a single object, i.e. consisting of connected segments. However, it was observed that in the 7-segment display the vertical segments do not appear to be connected unless the middle horizontal segment is lit. Figure 4(a) shows the edge map of digits 0,1 and 7 in which the the digits do not appear as a single region due to the above. Therefore, the edge map was pre-processed before skeletonizing to fill the gaps between the vertical segments. Since we are primarily concerned with the breaks between vertical segments, the pixels labelled X and Y are required to be non-edge pixels. The mask is centered on non-edge pixels and if the number of edge pixels in the top part of the mask ($P_0 - P_5$) and the bottom part of the mask ($P_6 - P_{11}$) both exceeds a particular threshold (in our application we use a threshold of 1) then the center pixel was flagged to be an edge pixel. This procedure is performed asynchronously, i.e. the breaks are not filled until all the pixels have been considered.Figure 4(c) shows the results of performing the filling on Figure 4. The skeleton of Figure 4(c) is shown in (d).

In step 4, we use Fourier descriptors to recognize the objects within the processing window. This step uses prototypes of all 10 digits, pre-stored in a data base, for classification. Objects not matching any of the 10 prototypes to a higher degree is classified as unknown. In addition to recognizing the objects we also determine the minimum enclosing rectangle of each of the object. This information is used in identifying the position of the object within the window. Also computed is the area of the object. This is required to discriminate between the digit 0 and 8 since the Fourier descriptors are unable to discriminate between them. At the end of this step all objects within the processing window have been identified as a digit or an unknown. This completes the first task of identifying the digits.

The next step is the formation of the decimal number from the individual digits identified within the window. In step 5, the objects within the window is processed from the rightmost object to the leftmost object using the above rules where necessary. The rightmost object is determined using the coordinates of the minimum enclosing rectangle.

Illustrated in Figure 4 are the sequence of processing steps for an 128X128 image. Since the recognition task is performed on edge maps of the image, for this procedure to perform error free, the edges needs to be generated correctly. This heavy dependence on the edge detector results limits the minimum size of the image. It was experimentally found that the LCD meter should occupy at least 15% of the total area to guarantee correct results.

4. Concluding Remarks

In this paper we describe development of modules associated with a robotic vision system for automatic inspection and manipulation tasks. The vision system consists of two groups of processing modules. The first comprises general purpose object recognition modules whereas the second comprises of specialized object status recogntion modules. Detailed development of modules for inspecting various types of meters, both analog and digital, is described.

REFERENCES

1. Ballard, D. H. and C. M. Brown, *Computer Vision*, Prentice-Hall, New Jersey, 1982, pp. 123-131.

ORIGINAL PAGE IS
OF POOR QUALITY

2. Hough, P. V. C., "Methods and Means for Recognizing Complex Patterns," U. S. Patent 3,069,654, Dec. 1962.

3. Kak, A. C. and J. S. Albus, "Sensors for Intelligent Robots," Handbook of Industrial Robotics, (S. Y. Nof, Editor), John Wiley & Sons, New York, 1985, pp. 214-230.

4. Persoon, Eric and King-Sun Fu, "Shape Discrimination Using Fourier Descriptors," *IEEE Transc. on Systems, Man, and Cybernetics*, Vol. SMC-7, No. 3, March 1977, pp.170-179.

5. Sanderson, A. C. and G. Perry, "Sensor-Based Robotic Assembly Systems: Research and Applications in Electronic Manufacturing," *Proceedings of the IEEE*, Vol. 71, No. 7, July 1983, pp. 856-871.

6. Trivedi, M. M., C. Chen, and S. Marapane, "A Vision System for Robotic Inspection and Manipulation," *Proc. of the Applications of Artificial Intelligence VI Conference*, SPIE Vol. 937, April 1988.

7. White, J. R., R. E. Eversole, K. A. Farnstron, H. W. Harvey, and H. L. Martin, "Evaluation of Robotic Inspection Systems at Nuclear Power Plants," NUREG/CR-3717, U. S. Nuclear Regulatory Commission, Washington, D.C., March 1984.

8. Zhang, T. Y. and C. Y. Suen, "A Fast Parallel Algorithm for Thinning Digital Patterns," *Communications of ACM*, Vol. 27, No. 3, March 1984, pp. 236-239.

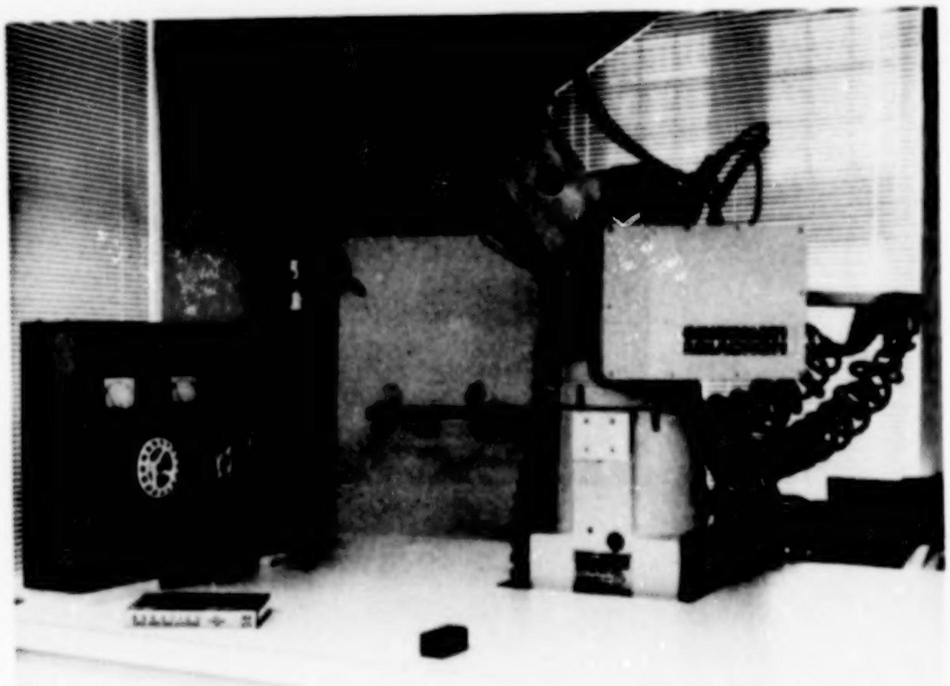


Figure 1: The test-panel and an industrial robot with vision, range, touch, force, and proximity sensory capabilities. The test-panel includes variety of displays, meters, valves, controls, and switches.

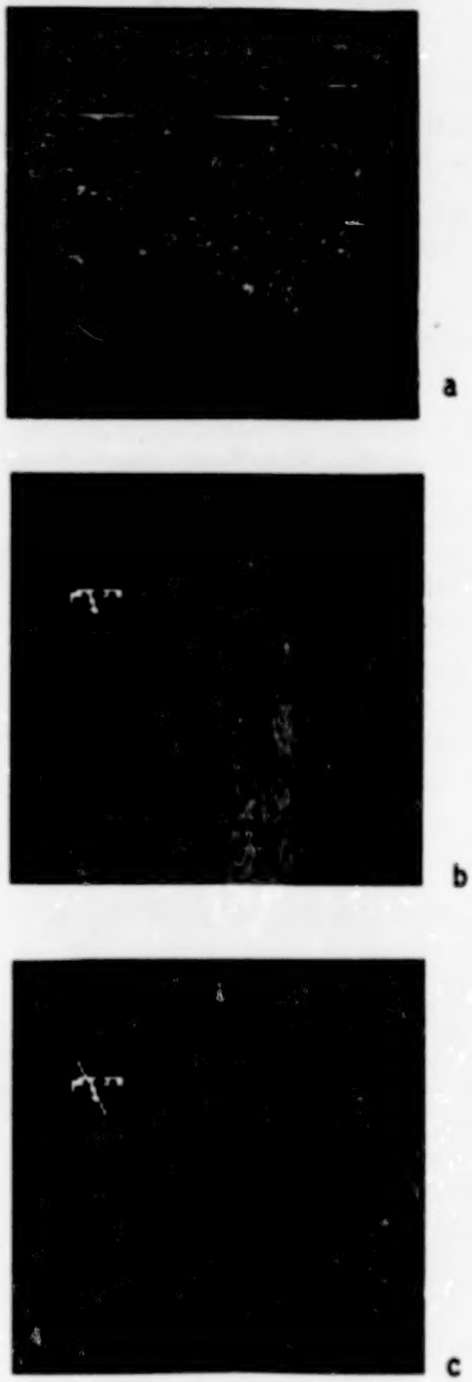


Figure 2: Sequence of processing steps. (a) original image, (b) edge map (9×9 mask) of (a), (c) detected needle superimposed on (b).

ORIGINAL PAGE IS
OF POOR QUALITY

ORIGINAL PAGE IS
OF POOR QUALITY

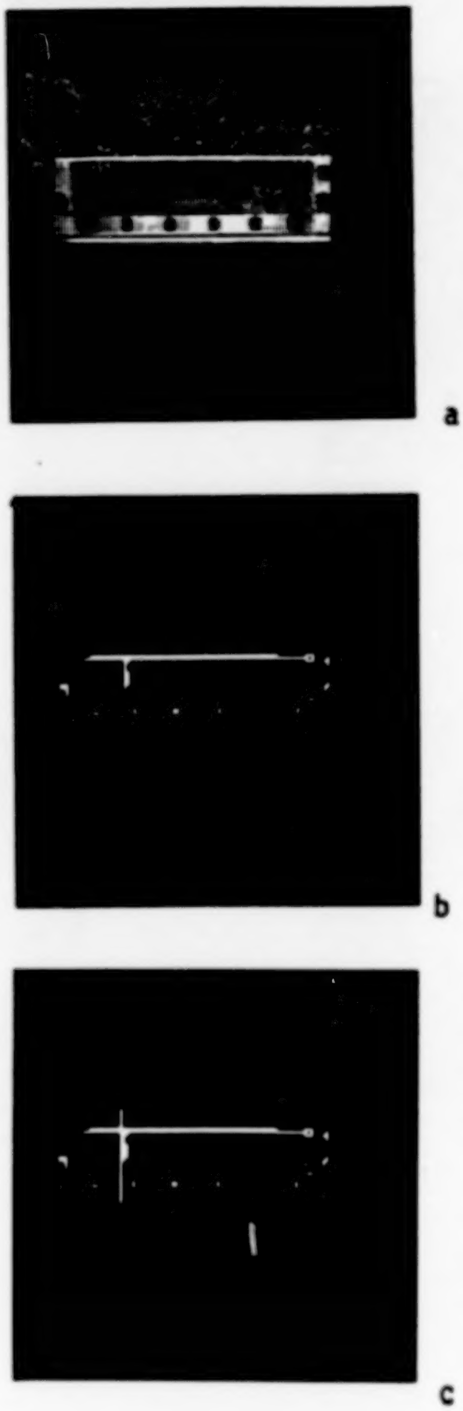
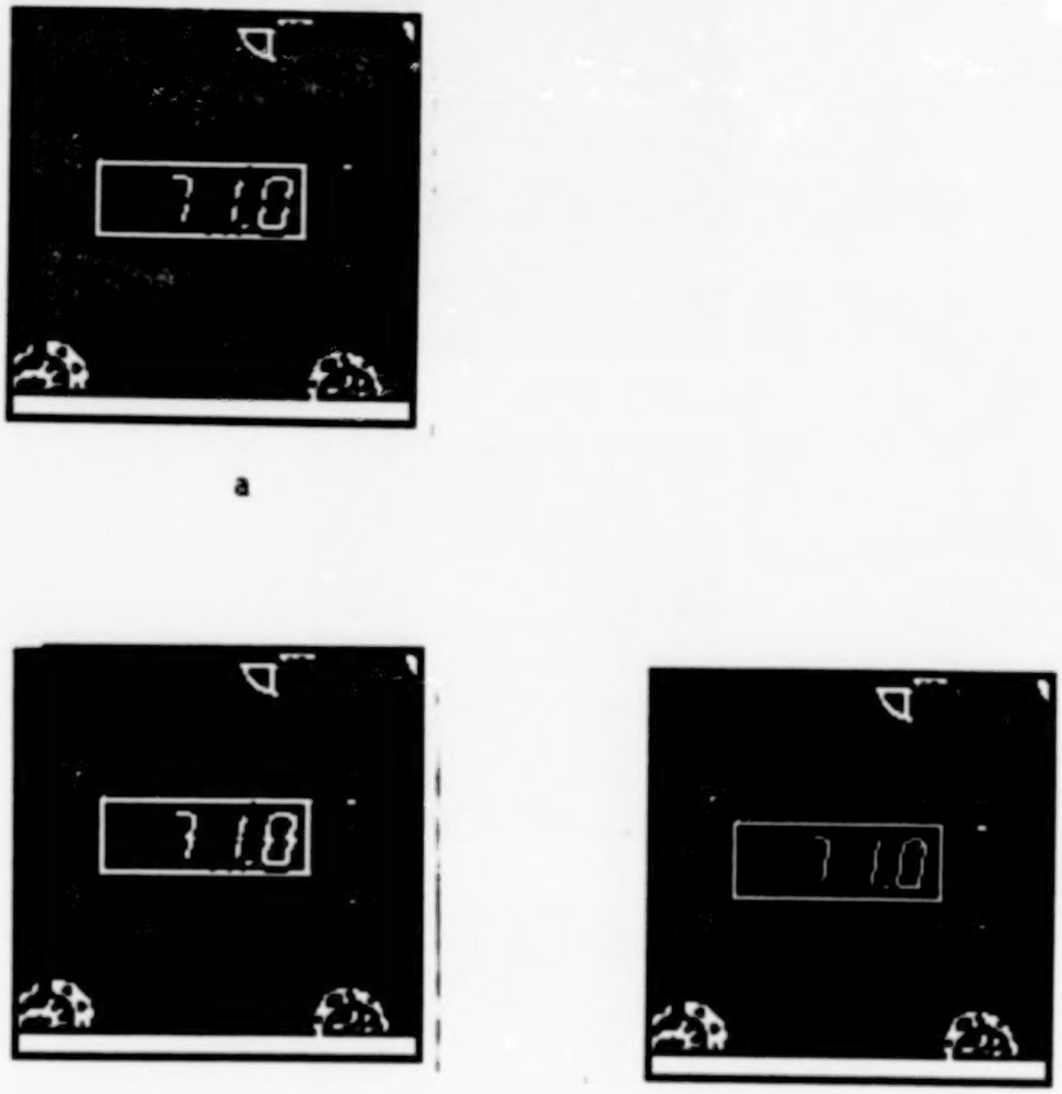


Figure 3: Sequence of processing steps. (a) original image. (b) edge map (5×5 mask) of (a), (c) detected needle superimposed on (b).

ORIGINAL PAGE IS
OF POOR QUALITY



a

b

c

Figure 4: Sequence of processing steps. (a) Typical edge maps of digits 0,1, and 7, (b) Results of filling, (c) Skeleton of (b)

N89-15583

KNOWLEDGE-BASED APPROACH TO SYSTEM INTEGRATION

W. Blokland, C. Krishnamurthy, C. Biegl and J. Sztipanovits
Department of Electrical Engineering
Vanderbilt University, Nashville, TN 37235

To solve complex problems one can often use the decomposition principle. A problem is however, seldom decomposable into completely independent subproblems. System integration deals with problem of resolving the interdependencies and the integration of the subsolutions. A natural method of decomposition is the hierarchical one. High-level specifications are broken down into lower level specifications until they can be transformed into solutions relatively easily. By automating the hierarchical decomposition and solution generation an integrated system is obtained in which the declaration of high level specifications is enough to solve the problem.

We offer a knowledge-based approach to integrate the development and building of control systems. The process modeling is supported by using graphic editors. The user selects and connects icons that represent subprocesses and might refer to prewritten programs. The graphical editor assists the user in selecting parameters for each subprocess and allows the testing of a specific configuration. Next, from the definitions created by the graphical editor, the actual control program is built. Fault-diagnosis routines are generated automatically as well. Since the user is not required to write program code and knowledge about the process is present in the development system, the user is not required to have expertise in many fields.

N89-15584

SUCCESSFUL EXPERT SYSTEMS FOR SPACE SHUTTLE PAYLOAD INTEGRATION

Keith Morris

Rockwell International
Space Transportation Systems Division
D282/900 FC43
12214 Lakewood Boulevard
Downey, California 90241

(213) 922-3700

ABSTRACT

Expert systems have been successfully applied to solve recurring NASA Space Shuttle orbiter payload integration problems. Recurrence of these payload integration problems is the result of each Space Shuttle mission being unique. The NASA Space Shuttle orbiter was designed to be extremely flexible in its ability to handle many types and combinations of satellites and experiments. This flexibility results in different and unique engineering resource requirements for each of the payload satellites and experiments. The first successful expert system to be applied to these problems was the Orbiter Payload Bay Cabling Expert System (EXCABL). It was developed at Rockwell International Space Transportation Systems Division. The operational version of EXCABL was delivered in 1986 and successfully solved the payload electrical support services cabling layout problem. As a result of this success, a second expert system, Expert Drawing Matching System (EXMATCH), was developed to generate a list of the reusable installation drawings available for each EXCABL solution. EXMATCH was delivered for operational usage in 1987. As a result of these initial successes, the need for a third expert system was defined and awaiting development. This new expert system, called Technical Order Listing Expert System (EXTOL), will generate a list of all the applicable reusable installation drawings available to support the total payload bay mission provisioning and installation effort. This paper describes these expert systems, the individual problems that they were designed to solve, their individual solutions, and the degree of success they have achieved. These expert systems' successes instantiate the applicability of this technology to the solution of real-world Space Shuttle payload integration problems.

INTRODUCTION

Rapid advancement of expert systems technology is contingent on wide-spread acceptance. To be widely accepted, expert systems must successfully provide needed solutions to existing real-world engineering problems. Providing examples of solutions to trivial generic problems does little to instantiate the applicability of expert systems technology to solve these nontrivial real-world problems. Providing examples of successful expert system application solutions to NASA Space Shuttle payload bay integration design problems does. The purpose of this paper is to disseminate knowledge of these successful applications, the problems that they solve, and the degree of success that they have achieved. Hopefully, this knowledge will be of some benefit to the expert system technology community as a whole and will play some small part in the advancement of this technology.

PRECEDING PAGE BLANK NOT FILMED

THE TOTAL PAYLOAD AND CARGO INTEGRATION AUTOMATION PROBLEM

The delivery of satellites and experiments into low-earth orbit by the Space Shuttle involves many preflight engineering planning, design, and integration tasks. These tasks include the following: selecting appropriate satellites and experiments to make up a mission payload set, locating each payload element within the payload bay, determining standard and unique services required by each payload, developing and documenting the payload to Space Shuttle orbiter interface requirements, selecting the individual cables necessary for providing the electrical services, preparing the electrical services cabling layout schematic, and preparing the technical instructions for mission payload installation and integration.

These tasks are carried out by teams of engineers, using both common and specialized engineering tools. Any changes in planning and design methods have to take the use of these existing tools into consideration. Because of their interdependence, the products of these teams are integrated into a master mission plan and schedule. Team technical support is supplied by highly trained experts who are rapidly reaching retirement age. Loss of an expert, is an undesirable event, not only with respect to the affected team's productivity, but also to the total payload integration task productivity as a whole.

Real-world space flight mission provisioning experience has shown that the ability to make mission manifest changes is mandatory. Other changes are to be anticipated because of further engineering analysis or design refinement. Changes caused by corrections of erroneous data, design omissions, and errors are also to be expected. Thus, change is a normal mode of operation and must be provided for, even close to launch time.

A major goal for all payload planning, design and integration tasks is to minimize this mission to mission change. This in turn will reduce paperwork, labor hours, and turnaround time. Standardization and automation are two powerful methods used in the payload integration process to minimize these changes.

In summation, Space Shuttle payload and cargo integration tasks are a collection of iterative interrelated activities, using specialized tools, responding to change, and led by vanishing experts. Automation, in order to be successful, must be tempered with these considerations. The following major objectives were established for each planning and design automation effort:

1. Reduce engineering labor hours

2. Retain technical expertise

3. Reduce end-to-end process time

4. Adapt to existing operating techniques and environment.

PAYLOAD BAY CABLING LAYOUT PLANNING AND DESIGN AUTOMATION

The Problem

The Space Shuttle payload cabling layout planning and design problem involves provision of the details required for the installation of cables to connect orbiter electrical services to the individual payload elements. Each Shuttle mission entails a different payload manifest, constituting a recurring planning and design problem. Mission payload manifest changes compound the problem further.

Standardized orbiter electrical services are provided through cables that connect the experiments and satellites to either the forward or aft orbiter payload bay bulkhead using standard electrical service panels. Cables are then routed from specific payloads to port and starboard standard interface panels. From these panels, cables travel to covered cabling trays for further routing to either the forward or aft bulkhead service panels. For efficiency, these cables are prefabricated and provided from a standardized inventory.

Because these standardized cables must service all payloads, regardless of their location within the payload bay, they are usually too long. The excess length of each cable must be dispositioned by forming a foldback or a loop (double foldback) within the routing tray. The trays are closed by covers that are located at designated locations along the tray. Cables with a diameter greater than 0.62 inches cannot be folded within the normal dimensions of the trays, because of radius bend constraints. Therefore, a special height appending foldback cover is required to replace the normal tray cover at the location of such a fold. Also, at the point where the cable leaves the tray to be routed to the interface panel or elsewhere, a special egress cover is required to replace the normal cover. Cables must also be separated by electromagnetic compatibility class through special channels provided in the routing trays.

Cabling installation practices are also governed by numerous other constraints and standard operating procedures. Based on heuristic knowledge, the above considerations, and the specific payload manifest, the cabling expert generates a hand drawn schematic that describes the cable routing solution. This schematic is subsequently used by a CAD/CAM specialist to produce a technical order (TO) schematic drawing of the cabling layout.

The Solution

The NASA Space Shuttle's payload bay cabling design task was the first automation problem to which expert systems technology was applied. An expert system, the EXCABL, was completed in September of 1986 and has been in operation since delivery. An overview of the EXCABL system is shown in Figure 1. The EXCABL system has completely automated payload bay cabling layout planning and design tasks. The cabling expert needs only to define the mission unique payload requirements

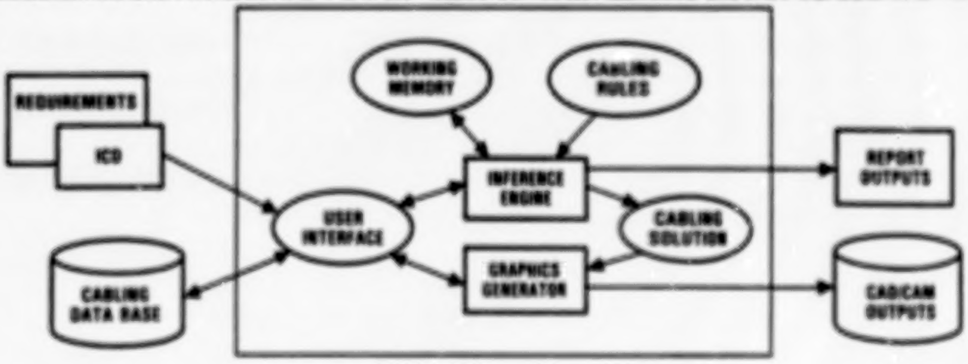


Figure 1. EXCABL System Overview

and constraints to generate the cabling solution CAD/CAM TO drawings and the printed reports. This was facilitated by the initial construction of a mission independent data base, containing all of the necessary payload bay hardware information required to perform Space Shuttle's cabling. The cabling experts' schematic solution is now automatically generated by the system as is the transfer of that information into CAD/CAM inputs.

All major automation objectives were met in the initial delivered system. The system has captured the required technical expertise and also provided a significant improvement in productivity. The cabling capabilities of EXCABL are such that a small percentage of actual cabling design tasks cannot be handled. Since the end product is a cabling installation drawing, any EXCABL solution can be manually modified or augmented to produce a more acceptable solution. The productivity improvement realized by this new capability is such that a typical mission cabling manifest, that formerly took a few labor intensive days for several cabling engineers, now takes only a few minutes.

The expert system portion of the operational version of EXCABL was implemented using Production Systems Technology's C-based version of OPS83 and the remaining portion was implemented using C. It is currently hosted on a CAD/CAM interfacing DEC MicroVax II system and integrated into the operational environment. The literature contains documentation of an early prototype version of EXCABL [Reference 2], problems associated with converting from a development system to a delivery system [Reference 3], and a case study of the development effort and lessons learned [Reference 1].

PAYLOAD BAY CABLE INSTALLATION TECHNICAL ORDERS

The Problem

The cabling layout solution schematic produced by EXCABL is only one of many Space Shuttle integration products necessary to accomplish the actual electrical services provisioning of its payload bay. Among the other products required are the installation configuration TO's for the cables and related hardware devices. These TO's contain the detailed instructions that are used by the payload integration crew to perform the actual cable and hardware device installation. The cabling installation TO's required for each flight are mission unique and dependent on the cabling solution generated by EXCABL.

To increase productivity, the concept of modularization was developed by the cabling design engineering group. This concept is to reuse previously generated TO's whenever possible, thereby eliminating the need to repeatedly redo labor-intensive documentation for the same installation. Implementation is accomplished by assigning basic TO numbers for each device, connector, or cable installation, and assigning dash numbers for the different configurations. If a needed TO does not exist, a new TO is generated, and a new dash number is assigned. This modularization, or reuse concept, is only made feasible by the standardization of cables, connectors, devices, and mounting positions, etc.

Identifying the set of all reusable cabling installation TO's for each given mission is a recurring integration problem. Since the set of cabling TO's for each mission is dependent on the EXCABL solution, any automation of this process must interface easily with existing EXCABL's outputs. Furthermore, maximum usage must be made of any intermediate information generated by EXCABL to support its final products. The desirability of integrating this process into the existing work environment, while cooperating with the EXCABL process, further constrains the design and development of any automated system solution.

Simply stated, the problem was to develop an automated system having the capability to identify and generate a list of all TO's required to perform the payload cabling installation for any Space Shuttle mission. The nonexistence of a required TO should be identified by the system to the user in order for the deficiency to be corrected and the process completed.

The Solution

There were two basic motivators for this project. First, was the demonstrated success of the EXCABL project. Second, was the practicality of automation based on the new concept of modularization and reuse. Furthermore, it was assumed that applying the experience and techniques gained from the EXCABL project would make this a low risk development effort [Reference 1]. Those assumptions proved to be correct in practice, and the entire development effort was straightforward and completed quickly.

An expert system called EXMATCH was placed in operational usage in January of 1988. The EXMATCH system has successfully automated the payload bay cabling installation TO generation task. Closely integrated with the EXCABL system, the cabling solution provided by EXCABL is automatically input to EXMATCH and a master listing of all required payload cabling installation TO's is generated. If a required TO does not currently exist, the system not only identifies this deficiency, but also identifies an existing similar drawing best match that may be modified to satisfy the deficiency.

To facilitate this system, an initial data base containing all current payload cabling installation TO numbers and data was constructed. For user convenience, the interface to maintain this data base was made an integral part of the EXCABL system. The development of the initial TO documentation and maintenance of the TO data base are the only functions not fully automated. Modifications to EXCABL were minimal. An overview of the integration of EXMATCH and EXCABL is shown in Figure 2.

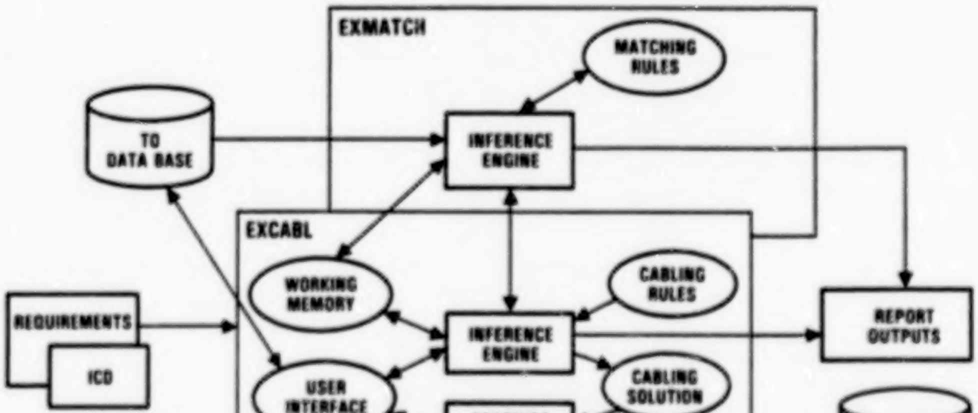


Figure 2. EXMATCH and EXCABL System Integration

The expert system portion of EXMATCH was implemented using Production Systems Technology's C-based version of OPS83. The remaining portion was implemented in DOD's registered trademarked language Ada. EXMATCH is currently cohosted with EXCABL on CAD/CAM interfacing DEC MicroVax II systems in the cabling design work place.

TECHNICAL ORDERS FOR TOTAL PAYLOAD INTEGRATION

The Problem

The Space Shuttle payload integration planning and design process culminates in the provision of a complete set of TO's containing the installation instructions needed to accomplish the total payload bay accommodation and installation task. To assist in the planning and installation process, a complete list of all applicable TO's for a mission is specified in a single document, the Mission Equipment Cargo Support Launch Site Installation (MECSLSI) drawing. Since each Space Shuttle mission is basically unique and design changes occur subsequent to initial payload manifesting, the identification of all required TO's for production of this drawing constitutes a continuing and complex integration problem.

Mission requirements are categorized as either mission common or mission unique. Mission common requirements are those requirements that once established, are standardized for all future missions. Mission unique requirements are dependent on each mission's objectives. Since each payload manifest is basically unique, the payload cabling layout schematic TO, produced by EXCABL is mission unique. However, it has been estimated that 90 percent of all mission requirements fall in the mission common category.

Based on flight requirements documentation, Interface Control Documents (ICD's), mission unique TO's, common TO's, and similarities to previous missions, etc., the MECSLSI development expert uses heuristic knowledge to generate the required drawing. If a design automation system could be developed to produce an initial MECSLSI containing only the mission common TO's labor requirements would be reduced considerably.

The Solution

A feasibility study was initiated in fiscal year 1987 to determine the practicality of developing an expert system to automate the production of the initial MECSLSI drawing. As a consequence of positive study results, it is expected that development of an expert system based design automation tool, EXTOL will be started in the near future. Not only will EXTOL produce the initial MECSLSI drawing but using heuristics and data from previous missions it will assist the user by producing a list of the best matches for nonexistent TO's in the mission unique category. If a close match cannot be found, it will identify that fact and provide further assistance to the user by presenting essential configuration information. It is expected that a working prototype could be quickly produced, and an operational system delivered shortly thereafter.

TOTAL PAYLOAD AND CARGO INTEGRATION AUTOMATION

EXCABL produces the mission unique cabling layout schematic TO product. EXMATCH uses information generated by EXCABL in its solution process to augment its knowledge and produce a list of all existing TO's that will be required to accomplish the cabling installation. If a required TO does not already exist, a best match or quite similar existing TO is identified as a baseline from which a new one can be constructed. When all of the required TO's are generated and this information is input to EXMATCH, it generates a complete list of cabling TO's. Together, these data will be furnished as electronic input's to the EXTOL system currently under consideration. These improvements should allow EXTOL to produce initial MECSLSI drawings that are over 90 percent complete. For TO's that are identified as needed, but not in existence, best match and configuration information, to greatly facilitate their generation, will also be produced.

EXCABL, EXMATCH and EXTOL constitute successful real-world demonstrations of the feasibility and benefits of applying expert systems technologies to the payload bay integration automation problem. Two of these successful systems have been integrated into the engineering work environment and cooperate to automate the overall payload integration management task. The third when completed will be integrated with the other two to further the goal of total payload bay integration automation. Since each expert system feeds its outputs directly to its successor, the productivity improvements of the group as a whole is greater than individual standalone systems could achieve. An overview of the integration management of these expert systems is shown in Figure 3. Using these systems as the core, other expert systems aimed at supporting the automation goal are in the concept development stage.

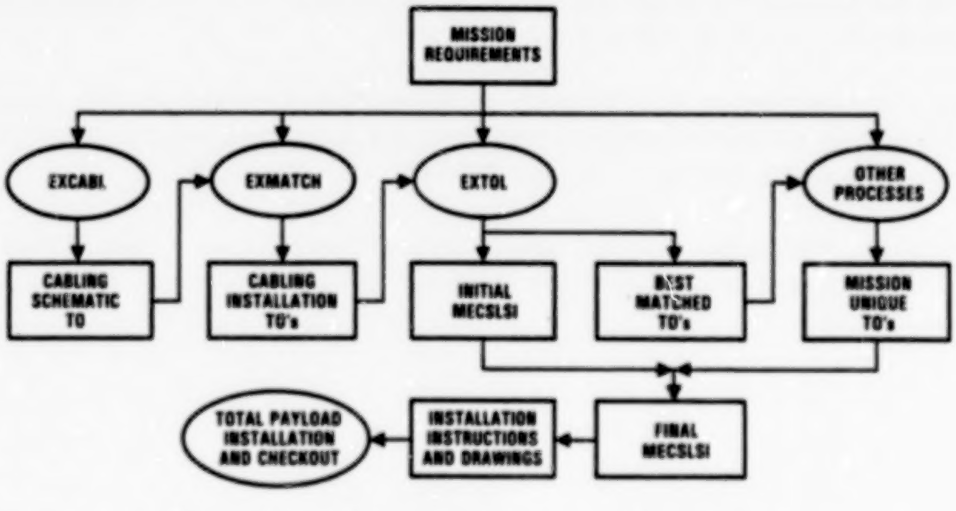


Figure 3. Payload and Cargo Integration Management

CONCLUSIONS

The high level of flexibility to handle diverse payloads provided by NASA's Space Shuttle orbiter presents present-day recurring payload and cargo integration problems. Expert systems technologies are being successfully applied to these real-world problems to provide automation where other approaches have failed.

EXCABL and EXMATCH are two highly successful examples of the applicability of using artificial intelligence techniques to solve these real-world integration automation problems. Both systems met the automation design objectives by reducing engineering-labor hours and end-to-end process time, capturing corporate technical planning and design expertise, and demonstrating that expert systems methods can successfully be integrated into existing operational systems and workplaces.

Successful development of a third expert system based integration automation tool is expected since much of the implementation methodology has already been successfully used by the EXMATCH system. These separate expert systems will work together to support overall payload integration management automation.

It is important to the advancement of expert systems technology that the knowledge of such successes as these be made public. Many successes are needed to instantiate the applicability of expert systems to solve real-world Space Shuttle payload integration problems.

ACKNOWLEDGMENTS

The author would like to thank Gil Nixon, Beshad Rejai, and Chuck Giffin for their contributions and support.

REFERENCES

1. Morris, K.E., Nixon, G.A. and Rejai, B. "EXCABL—Orbiter Payload Bay Cabling Expert System, A Case Study." *Proceedings of IEEE Westex-87 Expert Systems Conference*, Anaheim, California: IEEE, 1987, pp. 106-111

2. Saxon, C.R., and Schultz, R. "EXCABL: An Expert System for Space Shuttle Cabling." AI-85, *Proceedings of the First Artificial Intelligence and Advanced Computer Technology Conference*, Long Beach, California, 1985, pp. 127-140

3. Saxon, C.R. "Converting a Demonstration to a Delivery Expert System: Lessons From EXCABL." *Proceedings of IEEE Westex-86 Expert Systems Conference*, Anaheim, California: IEEE, 1986

N89-15585

AUTOMATED KNOWLEDGE BASE DEVELOPMENT FROM CAD/CAE DATABASES

R Glenn Wright and Mary Blanchard

PROSPECTIVE COMPUTER ANALYSTS, Inc.
1215 Jefferson Davis Highway, Suite 309
Arlington, VA 22202

ABSTRACT

Knowledge base development requires a substantial investment in time, money, and resources in order to capture the knowledge and information necessary for anything other than trivial applications. This paper addresses a means to integrate the design and knowledge base development processes through automated knowledge base development from CAD/CAE databases and files. Benefits of this approach include the development of a more efficient means of knowledge engineering, resulting in the timely creation of large knowledge based systems that are inherently free of error.

INTRODUCTION

Numerous problems traditionally associated with the development of knowledge based systems have been documented, including the availability of experts, the time required to build a system, unfamiliarity of the knowledge engineer with the domain, finding an expert who is enthusiastic about the project, etc.[1][3][10]. Prospective Computer Analysts, Inc, is investigating for NASA's Kennedy Space Center, methods to help resolve these and other problems through automated knowledge base development. Two of the methods and techniques for overcoming these problem areas, automated model building from CAD data and automated knowledge acquisition, are discussed. Each technique is used for generating different types of knowledge.

The automated model-builder generates the part of the knowledge base required for monitoring, control, and diagnosis of a system. The primary advantage associated with this method is a significant reduction in the amount of time and effort required to build a model representative of system connectivity and operational values, both normal and abnormal. Whenever the system design is changed, a new model can be generated quite easily. The knowledge used to generate a model can easily be extended to handle new parts and therefore new designs. Only the routines which directly interface with the CAD files need to be modified for other CAD packages and hardware.

Design knowledge capture techniques, beyond the standard documentation practices traditionally followed, are significantly more difficult to implement. For this particular application, we are referring to capturing design knowledge from the experts. Knowledge will be captured using techniques appropriate for the type of knowledge desired. The design knowledge is captured at the time when it is easiest for the designer to recall: during a design session. In order to diminish the problem of extracting implicit knowledge, indirect knowledge acquisition techniques can also be used.

Using either method, the knowledge is automatically documented and incorporated into the knowledge base in one step. The problem of knowledge verification and validation, however, still remains. This problem is reduced to some degree in the design knowledge capture process, which allow the designer to modify the knowledge base directly to correct any errors produced during it's creation. This capability provides the expert with a control capability over the knowledge base.

AUTOMATED MODEL BUILDING

By using knowledge about classes of components and design data contained in the CAD database, the generation of a model of a system being designed may be automated. This model can be used as part of the knowledge base for monitoring, control, and/or diagnostic software, as well as a communication tool between various people working on the project. For example, designers, test engineers, manufacturing and production personnel could all examine the same design (represented by the model) to check for inconsistencies and other factors throughout the life-cycle of the product. Examples of research in automated model building include that of Thomas [12], and University of Central Florida [6][7].

In order to deal with the vast amounts of information involved, the product being designed may be divided into hierarchial subsystems or modules. Each of these subsystems may be represented as a separate model with connections to the other models, thereby representing the subsystems to which it is connected. Each model would be contained in a separate knowledge base. As each subsystem is needed by the monitoring, control or diagnostic software, this knowledge base can be transferred into memory and the other written back out to disk.

By dealing with files produced from the CAD/CAE database instead of the entire database, the time required to produce a model and the amount of data handling can be reduced substantially. The CAD/CAE files would contain only the design data needed to generate the model for each subsystem, including: a unique name for each component, unique names for all connections between all components, standard nomenclature for each component, units used to measure output flow of a component, range of acceptable values associated with a component, part number (standard or manufacturers) for each component, and the tolerance associated with each measurement. All available measurements, commands, and components within the subsystem, are contained with the files. Information about the direction of flow included in the CAD/CAE database, and in turn the files, would increase the speed at which the model is built. Generally, the CAD/CAE database only indicates how components are physically connected, with no information given about the direction of the signal flow. As VHDL and

EDIF use becomes more widespread, this problem should lessen.

If different versions of a design are allowed, then different models representing these versions of the design must exist. When the designer feels significant changes have been made to a design, a new model must be generated. By requiring the designer to provide meaningful names for the models, including the version number of the design and for any connected subsystems, the model-builder will be able to handle multiple models for the same subsystem. The designer, or design configuration manager, should have the option of erasing models corresponding to old designs. However, the knowledge of the changes which occurred, and why they occurred, will be maintained in the design knowledge base. Neither the designers nor the support group will be allowed to erase the design knowledge base.

The model builder will use knowledge about the type and number of inputs and outputs associated with a component type to derive flow information. A connection list containing components and connection points, will initially be generated from one of the CAD files. The model builder algorithm will look for all the connections between the components listed in the connection list, then determine the input and output connections between the components. Additional information, such as typical or standard names for input and output connections, can be used. If the algorithm is unable to determine the direction of the signal flow for these components, based upon this information, the system will hold off making this decision until more information is known about the other components in the model. It is reasonable to assume that at some point the system will be able to make this decision for one pair of components in the model. Once the decision is made for one set, it narrows down the possibilities for the other components in the model, thus making it possible to determine the directional flow between pairs of components, which were previously eliminated.

Although the benefits of automated model building through this approach are many, certain problems which limit it's utility must be addressed. These problems include when the development of CAD designs is spread out over various, non-compatible, CAD/CAE hardware and software; handling the voluminous amount of information involved; constantly changing designs; the ability for many versions of the same design to exist; and for different designers to be using different versions of the same subsystem within their own design. These problems dictate the standardization of CAD/CAE design environments within common development and/or product lines. This will significantly reduce translation and configuration management requirements, and the resultant errors.

DESIGN KNOWLEDGE CAPTURE

CAD databases maintain the design representation and changes made to the design, however, no method currently exists to capture the reasons behind design decisions and changes. In order to capture this knowledge, it is necessary to supplement CAD/CAE software with a design knowledge capture tool.

By interacting with the designer using voice recognition and voice synthesis, the designer may be interviewed during the evolution of the design. Access to the design is provided through the model created from the CAD/CAE data. At the beginning of each design session, the design knowledge capture tool ensures that it has a model of the subsystem design to be worked on by the designer. If the model does not exist, one can be generated. It would not be practical to continually generate new models during the design session. Using this method, the design knowledge capture tool has an accurate model of the subsystem design at the beginning of the design session, and the designer is asked questions to determine what changes are being made.

Once a change is detected, questions can be asked to capture the designer's knowledge which went into making that change. This method leads the designer into explaining the design planning strategies, related analogies, general design knowledge, and the designers own experiences which went into making the design decisions. The knowledge used by an expert in designing must be represented using different data structures. For example, a plan can be best represented as a cyclic directed graph. The information required for each type of knowledge needs to be explicitly defined in order for questions to be generated. Associated with each question would be a set of expectations which can be compared with the answers received. Information received from the designer may or may not pertain to this question. Extraneous information received from the designer can still be processed by the system and incorporated into the knowledge base, however, the list of expectations will ensure that when an answer is given to the question, it will be recognized.

Typical questions asked could include:

1. (Why are you)(raising : changing : lowering : ...) (the) (pressure : temperature : dimensions : ...) (of the) (compressor : pump : power supply : ...) (?)

2. Why is this change necessary?

3. What other parts will be affected by this change?

4. How will these other parts be affected by this change?

5. Have you seen a similar configuration of parts previously?

Research was performed in learning casual models of physical mechanisms by understanding real-world natural language explanations of these mechanisms by the University of Connecticut [4][11].

Forward chaining rules can be used to select questions based upon the user's responses and related pieces of knowledge in memory. It is therefor very important to establish a relationship between what is being said by the designer and memory. When a human being is reading a sentence, the individual words are recognized, concepts are formed based upon those recognized words, and the user is reminded of related knowledge in memory. This should be the same process which takes place during natural language processing.

Intuitively it would appear a connectionist approach, such as that suggested by Jordan Pollack would be the best implementation for natural language processing [5][9]. This approach provides mechanisms for combining various types of knowledge for the processing of natural language. Each knowledge segment is represented as a node with excited or inhibited links to other nodes within the network. Each of the nodes represents a concept or microfeature within the domain. It allows domain and general world world knowledge and syntactic and semantic constraints to be integrated together for processing of the input.

Analogies, plans, experiences, general design frames, specific design frames, and design rules would be integrated into the network via nodal connections. This permits episodic memory to be used in the processing of the user's input. Nodal connections can be established between concepts or microfeatures within the network and slots of the general design frame for compressors, for example. Once the user mentioned the word "compressor", all of the related plans, experiences, etc. would be activated, in addition to the corresponding concepts and microfeatures within the network.

Some problems associated with this approach exist, however. Of primary concern is that the number of nodes and connections required is enormous. Either the nodes and connections for the entire network need to be represented in memory as they are, or a sub-network will need to be generated as words are processed. The first method will require a significant amount of memory. The second method involves additional overhead and therefore additional time. A parallel processor will be required to generate the new activation values for each of the nodes and update the weighted connections. The connections can be hardwired. Another question is how will the system handle the introduction of new nodes and the establishment of new connections. As each new part is mentioned by the user, it should become a part of the network. It will also be difficult to relate the activated concepts/microfeatures within the network to the changes required in the plans to reflect this new information. It will be difficult to incorporate information received from the designer into the plans, analogies, etc. used to represent the design knowledge.

Another obvious problem is the cost of a parallel processor. It would not be feasible to provide every design engineer with a massively parallel processor to perform natural language processing. An alternative solution is to combine a conceptual analyzer [2] where syntactic and semantic information can be stored, with episodic memory, represented as general design frames, specific design frames, design rules, plans, analogies, and experiences. Episodic memory will be stored in terms of a type of component, a specific component, and/or a characteristic of the component. Activation of a related concept in the dictionary (used in conceptual analysis) would cause the activation of related pieces of episodic memory. This approach will allow syntactic and semantic constraints, domain and general world knowledge with episodic memory to be integrated, although not to the same degree as a connectionist network. Also, the speed of processing will be slower. However, at the present time this method is feasible, while the connectionist approach should be considered appropriate for future applications.

Design Knowledge Capture Considerations

A key element in design knowledge capture is the need to be able to recognize changes in the focus of attention. When the designer changes the course of the conversation, new questions and expectations need to be generated. The currently active knowledge in memory needs to be changed to reflect the new focus of attention. Consideration must be given as to whether to incorporate all of the previous information given prior to the change in focus into the knowledge base, erasing the related questions and expectations, or to keep this information on the chance that the designer will refer to an earlier topic. This will require considerable overhead and the need for questions to be generated to determine which one of the previous subjects the designer is discussing. One possible solution is to consider one subject at a time and look upon any diversion as a change in focus.

Restricting the designers' ability to modify design information given within certain time periods, e.g. daily, weekly, etc., but allowing visibility to previous data is desirable. One possibility is to allow the designer to change knowledge given during the current (i.e., present day) design session, but provide read-only access to knowledge given in prior design sessions. This protects the knowledge of other designers contained within the knowledge base, and prevents the loss of previous knowledge which would be very difficult to replace in the event of accidental loss. Further restrictions with respect to the amount of knowledge which can be modified in the same subject area would also be warranted for the same reason. A designer may also add knowledge to the knowledge base using techniques discussed in the following paragraphs.

Knowledge acquisition techniques fall into one of two categories, direct or indirect [8]. Direct techniques include: interviews, questionnaires, observation of task performance, protocol analysis, interruption analysis, closed curves and inferential flow analysis. Questionnaires can be generated for holes in the knowledge base and the user asked to fill out these questionnaires. Drawing closed curves can be used to help discover analogies. The designer would be asked to draw a closed circle around related objects. Next, questions can be generated to determine the similarities between the objects and derive design rules which can be extended from one domain to another.

The indirect methods include multidimensional scaling, hierarchical clustering, general weighted networks, ordered trees, and repertory grid analysis. General weighted networks can be used to discover planning strategies. The network is made up of concepts represented as nodes. The links are used to show order and direction of the steps in the plan. In this case, the concepts are steps of a plan. The user would be given major steps within a plan and through questions, the substeps of the plan would be discovered. The designer would establish the order of the substeps using links to connect the nodes.

The total memory of the system includes a model of the system designed and the design knowledge which went into the design. However, additional knowledge can be added such as, repair and maintenance data on the parts contained within the system, manufacturing knowledge on how to produce the parts, the list of manufacturing equipment available and their capabilities and limitations, and knowledge about the environment in which the system will be operating. An additional layer can be added which would act as a communication module, permitting people from various departments access to information about the system. By allowing this open exchange of information during the design of the system, the probability of producing a system which can be manufactured and meet operational requirements, the first time, is increased significantly. This knowledge should be available and easily accessible throughout the life cycle of the system. It can be used when a design revision, a new environment for the system, or when a change in manufacturing equipment is being considered.

Other knowledge acquisition modules would need to be developed to deal with these other domains. Also a suitable network of hardware and software would need to be selected. Other factors to consider would be, an increase in security, how often will the knowledge bases be accessed, physical locations of people accessing the information.

CONCLUSIONS

Two methods for model building and design knowledge capture for automated knowledge base development have been presented. Current technology provides the means to address this topic and initiate development with meaningful results which may be applied towards solving many design knowledge capture and knowledge base development problems which exist today.

The means to overcome limitations in today's technology is available, however long term solutions would greatly benefit from connectionist methodologies utilizing massively parallel processing in a standardized CAD/CAE development environment.

REFERENCES

[1] Berry, Diane C., "The Problem of Implicit Knowledge", Expert Systems, August 1987, Vol. 4, No. 3

[2] Gershman, Anatole V., "A Framework for Conceptual Analyzers", Chapter 6, pp 177-197, in Strategies For Natural Language Processing edited by Wendy G. Lehnert and Martin H. Ringle, Lawrence Erlbaum Associates, 1982

[3] Hoffman, Robert R., "The Problem of Extracting the Knowledge of Experts from the Perspective of Experimental Psychology", AI Magazine, Summer 1987

[4] Klimczak, Benjamin "Learning Casual Models of Physical Devices by Understanding Real-World Explanations using a Global Dictionary", Technical Report #86-23, December 1986, University of Connecticut, Storrs, CT 06268

[5] Lehnert, Wendy G., "Knowledge-Based Natural Language Understanding: A AAAI-87 Survey Talk", COINS Technical Report 88-02, University of Massachusetts, Amhert, Massachusetts

[6] Dr. Myler, H.R. and Dr. Gonzalez, A.J., "NASA Research Grant, Automated Knowledge Generation First Quarterly Review Meeting", Computer Engineering Department, University of Central Florida, Orlando, Florida

[7] Dr. Myler, H.R. and Dr. Gonzales, A.J., "NASA Research Grant, Automated Knowledge Generation, Second Quarterly Review Meeting", Computer Engineering Department, University of Central Florida, Orlando, Florida

[8] Olson, Judith Reitman and Rueter, Henry H. "Extracting Expertise from Experts: Methods for Knowledge Acquisition, Expert Systems, August 1987, Vol. 4, No. 3

[9] Pollack, Jordan B., "On Connectionist Models of Natural Language Processing", MCCS-87-100, New Mexico State University, Las Cruces, New Mexico

[10] Prerau, David S., "Knowledge Acquisition in the Development of a Large Expert System", AI Magazine, Summer 1987

[11] Selfridge, Mallory and Cuthill, Barbara, "Automatic Invention: Inferring the Casual Model of a Physical Mechanism From a Description of Desired Behavior", Technical Report #86-17, December 1986, University of Connecticut, Storrs, CT 06268

[12] Thomas, Stan J. "NASA/ASEE Summer Faculty Research Fellowship Program, Automated Construction of a Knowledge Base From Computer Aided Design Data", Wake Forest University

N89-15586

Dynamic Reasoning in a Knowledge-based System

Anand S. Rao and *Norman Y. Foo*
Department of Computer Science
University of Sydney, NSW-2006
Australia

anand@basser.oz and norman@basser.oz

Abstract

Any space based system, whether it is a robot arm assembling parts in the space or an onboard system monitoring the space station, has to react to *changes* which cannot be foreseen while on earth. As a result, apart from having domain-specific knowledge as in current expert systems, a space based AI system should also have *general principles of change*. This paper presents a modal logic which can not only represent change but also reason with it. Three primitive operations, *expansion*, *contraction* and *revision* are introduced and axioms which specify how the knowledge base should change when the external world changes are also specified. Accordingly the notion of *dynamic reasoning* is introduced, which unlike the existing forms of reasoning, provide general principles of change. Dynamic reasoning is based on two main principles, namely *minimize change* and *maximize coherence*. A possible-world semantics which incorporates the above two principles is also discussed. The paper concludes by discussing how the dynamic reasoning system can be used to specify actions and hence form an integral part of an autonomous reasoning and planning system.

1. Introduction

Due to the prohibitive costs of manned missions to outer space, unmanned explorations of distant objects is the only credible alternative. As man starts exploring deeper into space, the need for advanced *autonomous systems* becomes inevitable. Space based systems need to be autonomous with respect to two important aspects, namely, a) control and b) reasoning. As the communication delay between autonomous vehicles in outer space and earth is beyond acceptable limits, such vehicles should have independent control of their movement. For much the same reasons, they should also have independent decision making skills. Unless they are equipped with such capabilities they are unlikely to survive in an hostile environment. Also it is very difficult, if not impossible to preprogram such capabilities. This paper discusses only the reasoning capabilities of an autonomous system and not its control aspects.

Two main requirements for an autonomous reasoning system are

a. that the system be *dynamic*, i.e. any changes in the external environment should be immediately reflected in the systems view of the environment. In other words the system should be capable of *changing* its *knowledge-base* automatically based on the changes in the external environment.

b. that the system be *reactive* [8] i.e. it should be capable of changing its focus and pursuing an alternative goal if and when it is required. In other words the system should be capable of *changing* its *goals* automatically based on the changes in the external environment.

An example would help to illustrate the dynamic and reactive aspects of an autonomous, space based AI system. Consider a robot whose main goal is to explore the terrain of an hostile environment. During the exploration phase the sensors of the robot will constantly keep feeding information about the external environment. Some of this information might be in conflict with the robot's existing knowledge. The responsibility of the *dynamic reasoning* component of the robot is to accommodate this information from the external world with as little damage as possible to the current knowledge base. During the exploration phase if there is a sudden unexpected event, say a volcanic eruption, then the robot must be capable of *reacting* to this situation by abandoning its goal of exploration and instead acquire the goal of survival and try to achieve it. This process of changing the focus of attention or changing the goals is an integral part of the *reactive* component of the reasoning system. In [8] and [7] Georgeff et. al. describe the Procedural Reasoning System (PRS) and discuss its use in an autonomous mobile robot and in a diagnostic system for the space shuttle.

C-4 *

It is quite clear that any space based system, whether it is a robot arm assembling parts in the space or an onboard system monitoring the space station, it has to react to *changes* which cannot be foreseen while on earth. As a result, apart from having domain-specific knowledge as in current expert systems, a space based AI system should also have *general principles of change*. Unfortunately, standard first-order logic and deductive reasoning have very little to say about dynamic reasoning. This is because most of the reasoning which is done using first-order logic has been restricted to computing the logical consequences of sentences, which can be categorized as *static reasoning*. This paper describes a modal logic and illustrates how dynamic reasoning can be carried out in this logic. The dynamic reasoning described in this paper incorporates two general, intuitive principles of change, namely, *minimize change* and *maximize coherence*. The axiomatization of this dynamic reasoning system is inspired by the postulates of theory change, first proposed by Gardenfors et. al. [6, 1]. Although, most of this paper will concentrate on the dynamic reasoning system, it also discusses how to integrate the dynamic reasoning system with a reactive planning system, to obtain a full-fledged embedded system capable of reasoning and planning autonomously in any hostile environment.

2. Statics of Belief Systems

Traditional knowledge-based systems, typically have two main modules:

a. Knowledge Base (KB) - which contains domain specific information in some formal language, which is normally first-order logic or a syntactic variant of first-order logic with procedural attachments.

b. Inference Engine (IE) - which performs *static* reasoning on the knowledge base.

The KB is taken to be a set of beliefs about a particular problem domain and the IE is capable of answering queries regarding the belief system. Such knowledge bases are essentially static, as they cannot represent or reason about how the KB changes.

This section will present a KB or belief system, which can model the evolving nature of knowledge bases. The static aspects of the belief system will be dealt in this section and the dynamic aspects of the belief system will be postponed to the next section.

The formal language $\mathcal{L}_1$ under consideration is a modal logic of beliefs. The objects of beliefs will be taken to be first-order formulas with equality [14]. The beliefs will be taken to be time-dependent. Thus the formula `BELIEF(t, ϕ)` represents the belief of the agent (or robot or the AI system) at time point `t`, that the formula ϕ is the case. For example, the formula `BELIEF(13:00, power_left(2))`, might be the belief of the robot that at 13:00 hrs, the number of hours of power left for it is 2 hours. In the language $\mathcal{L}_1$, quantifying temporal terms into the scope of the modal operator `BELIEF` is allowed, but quantifying individual terms is not allowed.

Usually the semantics of the `BELIEF` operator is given in terms of a possible-world accessibility relation, $\mathcal{B}$, which maps a possible world to a set of possible worlds. In the language $\mathcal{L}_1$ the relation $\mathcal{B}$, maps a possible world *at a given instant of time* to a set of possible worlds *at that given instant of time*. Also the satisfaction is with respect to a world *at a particular instant of time*. Thus if KI, is a Kripke interpretation of modal logic, and TA the term assignment, then the satisfaction of belief formulas with a particular variable assignment VA, is given as follows,

KI, w, TA(`t`) $\models$ `BELIEF(t, ϕ)`[VA] iff for all w′ such that $\mathcal{B}$(w, TA(`t`), w′), KI, w′, t $\models$ ϕ[VA].

The axiomatization for the above time-dependent belief system, called the B-modal system, are the axioms of first-order temporal logic and the standard KD45 axiomatization for beliefs [10, 17]. In a KD45-modal system the $\mathcal{B}$ relation has to be serial, transitive and euclidean. The class of models of the time-dependent belief system, whose $\mathcal{B}$ relation satisfies the above conditions are called $\mathcal{B}$-models. The soundness and completeness of the time-dependent belief system can be stated as follows,

Theorem 2.1: The B-modal system is sound and complete with respect to the class of $\mathcal{B}$-models.

Proof: The proof of the above theorem is straightforward and is proved in [17].

The `BELIEF` modal operator can also be treated as a self-belief operator as in autoepistemic (AE) logic [15]. In AE logic the agent reasons about his own beliefs and lack of beliefs. An agent believes a sentence if it is contained in his set of beliefs at that current instant of time. He does not believe it, if it is not contained in his set of beliefs. Thus AE logic is complete with respect to the beliefs and non-beliefs of the agent. A Kripke interpretation where the beliefs are treated as self-belief operators will be called an autoepistemic Kripke interpretation or AKI.

3. Semantics of Dynamic Belief Systems

In most of the commercially available knowledge-based systems or expert system shells, any changes to the KB have to be done manually. For a space-based AI system this solution is unacceptable for two main reasons,

1. the time delay for updates will be too costly and might endanger the mission and
2. maintaining the integrity of the KB will be difficult, especially if the input data is inconsistent with the existing KB.

Thus for any space-based AI system, automatic updates of the KB is a must. Two of the most popular systems for updating knowledge bases are the Truth Maintenance System (TMS) [4] and Assumption-based TMS (ATMS) [11]. TMS keeps track of how each and every formula was inferred, which are called *justifications* and incrementally modifies these justifications whenever there is an update. ATMS keeps track of the premises which are used in deriving a formula, called *assumptions* , and incrementally modifies them. However, in both these mechanisms the book-keeping required may not only be space consuming, but might also turn out to be time-consuming. Thus for a space-based AI system such mechanisms may be unsuitable.

In this section three basic *dynamic* operations, namely, *expansion*, *contraction* and *revision* are introduced. These operators are then used for reasoning about changing knowledge bases. The semantics of these operations are discussed in this section and the dynamic reasoning system is discussed in the next section.

Consider the situation where the agent (or robot) receives a first-order sentence ϕ from the external world. The relationships between this formula and the set of beliefs of the system at time `t`, can be enumerated as follows,

1. the agent believes in ϕ at `t`.
2. the agent believes in $\neg\phi$ at `t`.
3. the agent does not believe in ϕ nor $\neg\phi$ at `t` and hence the agent is *agnostic* about ϕ.
4. the agent does believes in ϕ and $\neg\phi$ at `t` and hence the agent has *inconsistent* beliefs about ϕ.

Inconsistent belief states are disallowed and hence, relation 4) above does not hold. The process by which an agent moves from one of the three regions(1, 2, or 3) to any of the other two is called the *dynamics* of the system. As there are three states and each transition involves two states there are totally 3^2 different transitions. Ignoring the trivial transitions of remaining in the same state six different transitions are left.

- **Expansion:** Agnostic state → Belief in ϕ.
- **Contraction:** Belief in ϕ → Agnostic state.
- **Revision:** Belief in $\neg\phi$ → Belief in ϕ.
- **N-Expansion:** Agnostic state → Belief in $\neg\phi$
- **N-Contraction:** Belief in $\neg\phi$ → Agnostic state
- **N-Revision:** Belief in ϕ → belief in $\neg\phi$.

The terminology and approach is an extension of the work done by Gardenfors and others [6, 1]. While their approach is non-modal and at the meta-level, we introduce modal operators and carry out the analysis at the object level.

The language of the time-dependent belief system is extended to the language L_2, by introducing three *dynamic modal operators-* `EXPAND`, `CONTRACT` and `REVISE` to denote expansion, contraction and revision respectively. Thus `EXPAND(t, ϕ, u)` is read as 'the expansion by the agent at `t` with respect to ϕ is `u`', where ϕ is a first-order sentence and `t` and `u` are temporal terms (constants or variables). There is no need for any modal operators for N-expansion and N-contraction as they can be expressed by the modal operators `EXPAND` and `CONTRACT` respectively. Strictly speaking there is no need for the modal operator `REVISE` also as it can be defined using `EXPAND` and `CONTRACT`. Nesting of these operators are not allowed. As AE belief systems are uniquely determined by the first-order formulas, it is sufficient to consider the expansion, contraction and revision of first-order sentences alone. For example, expanding with respect to a belief sentence in an AE belief system is equivalent to expanding with respect to the object of the belief, and expanding with respect to a non-belief sentence is equivalent to contracting with respect to the object of the non-belief.

Given a set of beliefs at some particular instant of time `t`, and the nature of change, namely, expansion, contraction or revision the set of beliefs at the next time instant can be constructed and the semantics of the modal operators `EXPAND`, `CONTRACT` and `REVISE` are definable using this construction [17]. Alternatively, the semantics of the dynamic modal operators, can be given based on the autoepistemic Kripke interpretation, AKI. The latter approach is followed in this paper. The semantics of dynamic operators are based on *selection functions*, which select some possible worlds as being *closer* to the current world than the others [2, 12]. When the agent performs expansion or contraction, he is said to move into one of these closer worlds and designate these worlds as the worlds of the next time instant(s).

Consider an agent with a proposition `p` in some world `w` at time `t1`. Now the agent wants to expand with respect to the formula `r`. He can do so in many ways. He can move into a time point `t2`, where `p`, `r` are true or a time point `t3` where only `r` is true or a time point `t4` where `p`, `q`, `r` are true and so on. Amongst the different alternatives the agent should *choose* only some of them, based on certain criteria. The principles of minimal change and maximal coherence will be used in selecting the alternatives. According to these criteria, the selection function for expansion, or *expansion function*, denoted by $\mathcal{E}$, should choose the time point `t2`, where `p`, `r` are true. The other time points are also accessible from `t1` but all of them have to pass through the time point `t2`. Thus the time point `t3` can be obtained from `t2` after a contraction with respect to `p`. Similarly, the time point `t4` can be obtained from `t2` after an expansion with respect to `q`. Also it should be noted that the expansion function $\mathcal{E}$ and the contraction function $\mathcal{C}$, depend on the current time point, the current world and the proposition with respect to which the expansion/contraction is performed. The result of the selection function is a set of time points.

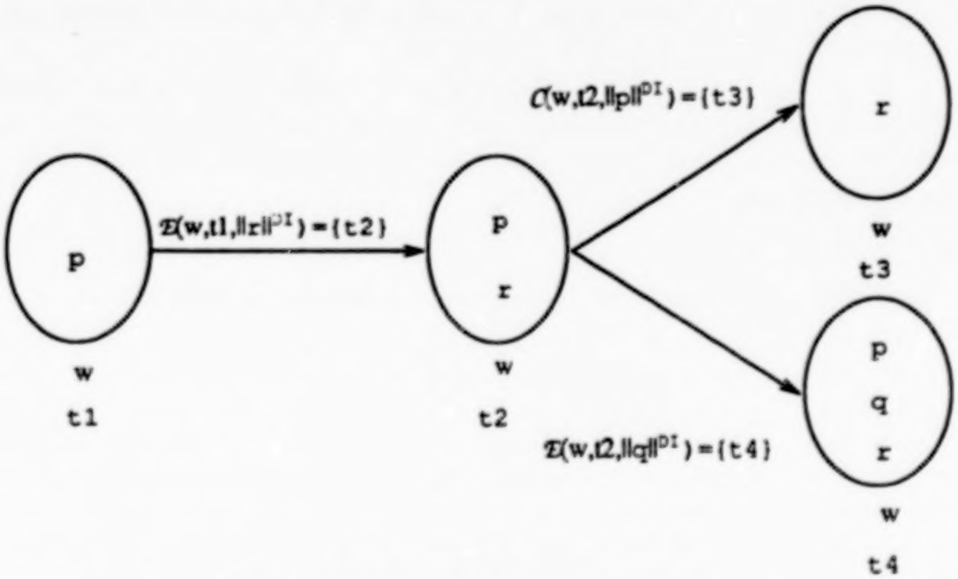

$\mathcal{C}(w, t2, \|p\|^{DI}) = \{t3\}$
$\mathcal{E}(w, t1, \|r\|^{DI}) = \{t2\}$
$\mathcal{E}(w, t2, \|q\|^{DI}) = \{t4\}$
p
w
t1
p
r
w
t2
r
w
t3
p
q
r
w
t4

Figure 1. Expansion and Contraction Functions

Figure 1 shows the possible relationships between the different time points.

The selection functions discussed above can be generalized to handle first-order sentences, instead of just propositional formulas. The semantics of the dynamic belief system is given below.
Definition: The *dynamic interpretation* is a tuple, D = <$\mathcal{E}$, $\mathcal{C}$, AKI>, where

- $\mathcal{E}$ and $\mathcal{C}$ are functions which map, W (set of worlds), TU (set of time points), and 2^{2^W} (set of set of worlds) to a set of time points or 2^{TU}.
- AKI is an autoepistemic Kripke interpretation.

The satisfaction of belief formulas are as described before. Satisfiability of the dynamic modal operators is as follows,
Definition: A dynamic interpretation DI = <$\mathcal{E}$, $\mathcal{C}$, AKI>, satisfies a well formed formula ϕ, at world w and time `t`(written as DI, w, `t` $\models \phi$) given the following conditions,

1. $\mathrm{DI}, w, t \models \mathtt{EXPAND}(t, \phi, u)$ iff $u \in \mathcal{E}(w, t, \|\phi\|^{\mathrm{DI}})$, where ϕ is a first-order sentence.
2. $\mathrm{DI}, w, t \models \mathtt{CONTRACT}(t, \phi, u)$ iff $u \in \mathcal{C}(w, t, \|\phi\|^{\mathrm{DI}})$, where ϕ is a first-order sentence.
3. $\mathrm{DI}, w, t \models \mathtt{REVISE}(t, \phi, u)$ iff $\mathrm{DI}, w, t \models \mathtt{CONTRACT}(t, \neg\phi, v)$ and $\mathrm{DI}, w, v \models \mathtt{EXPAND}(v, \phi, u)$, where ϕ is a first-order sentence.

The notation $\|\phi\|^{\mathrm{DI}}$ stands for all the worlds of the interpretation DI, which satisfies ϕ. More formally, $\|\phi\|^{\mathrm{DI}} = \{w \mid \mathrm{DI}, w, t \models \phi \text{ for any } t \in \mathrm{TU}\}$. Condition 3 is an analog of the Levi identity, which is a theorem in Alchourron, Gardenfors and Makinson [1].

This completes the semantics of the dynamic operations. In the next section the axiomatization of the dynamic belief system is discussed. This axiomatization forms the basis of the dynamic reasoning system.

4. Dynamic Reasoning System

A reasoning system is specified by a set of axioms and inference rules. Given a set of formulas, the axioms and inference rules are used to infer more formulas which are called the logical consequences of the axiom system. In the case of static reasoning the formulas are first-order formulas which represent the world at the current time instant. Using the axioms and inference rules of first-order logic more formulas can be derived which also represent the world at the current instant of time. In the case of dynamic reasoning the formulas represent the world of the current time instant and the formulas which are added to the existing world or removed from the existing world. Based on the axioms and inference rules of the dynamic reasoning system more formulas are derived, which represent the state of the world at the *next* time instant. Thus the dynamic reasoning system will determine what will be true in the next time point, given the current time point and the nature of change. The next three subsections describe the axioms and inference rules of the dynamic reasoning system.

4.1 Expansion

By the very definition of expansion, the expansion of a FOE sentence ϕ, should result in a belief of ϕ at the next time instant. This is called the *Axiom of Inclusion* [6, 1].

Axiom of Inclusion

(AE1) $\mathtt{EXPAND}(t, \phi, u) \rightarrow \mathtt{BELIEF}(u, \phi)$

Semantically the above axiom states that,

(CE1) If $u \in \mathcal{E}(w, t, \|\phi\|^{\mathrm{DI}})$ then $w \in \|\mathtt{BELIEF}(u, \phi)\|^{\mathrm{DI}}$.

Expansion is an operation which preserves the beliefs of an agent. If u is the world obtained after expanding t with respect to ϕ then all the belief formulas at t continue to hold at u although some of the non-belief formulas at t may be believed at u. In other words, expansion converts some of the agent's non-beliefs into beliefs, while at the same time preserving his existing beliefs. This property is called the *Axiom of Preservation of Beliefs*.

Axiom of Preservation of Beliefs

(AE2) $\mathtt{EXPAND}(t, \phi, u) \rightarrow$
$(\mathtt{BELIEF}(t, \alpha) \rightarrow \mathtt{BELIEF}(u, \alpha))$

Semantically, the above axiom states that,

(CE2) If $u \in \mathcal{E}(w, t, \|\phi\|^{\mathrm{DI}})$ and $w \in \|\mathtt{BELIEF}(t, \alpha)\|^{\mathrm{DI}}$, then $w \in \|\mathtt{BELIEF}(u, \alpha)\|^{\mathrm{DI}}$.

If the agent already believes in ϕ at t then expansion with respect to ϕ will yield no new beliefs. This is a case of trivial expansion where the agent's beliefs as well as non-beliefs are preserved. This is called the *Axiom of Trivial Expansion*.

Axiom of Trivial Expansion

(AE3) $\mathtt{BELIEF}(t, \phi) \wedge \mathtt{EXPAND}(t, \phi, u) \rightarrow (\mathtt{BELIEF}(t, \alpha) \equiv \mathtt{BELIEF}(u, \alpha))$.

The semantic condition for the above axiom states is as follows,

(CE3) If $w \in \|\mathtt{BELIEF}(t, \phi)\|^{\mathrm{DI}}$ and $u \in \mathcal{E}(w, t, \|\phi\|^{\mathrm{DI}})$ then ($w \in \|\mathtt{BELIEF}(t, \alpha)\|^{\mathrm{DI}}$, iff $w \in \|\mathtt{BELIEF}(u, \alpha)\|^{\mathrm{DI}}$).

The operation of expansion is monotonic. Thus when belief in world t implies belief in world v then belief in the expanded world of t will imply belief in the expanded world of v. The following *Axiom of Monotonicity* mirrors this fact.

Axiom of Monotonicity

(AE4) $(\mathtt{BELIEF}(t, \alpha) \rightarrow \mathtt{BELIEF}(v, \alpha)) \wedge \mathtt{EXPAND}(t, \phi, u) \wedge \mathtt{EXPAND}(v, \phi, y) \rightarrow (\mathtt{BELIEF}(u, \beta) \rightarrow \mathtt{BELIEF}(y, \beta))$.

Semantically the above axiom translates into the following condition.

(CE4) (If $w \in \|\mathtt{BELIEF}(t, \alpha)\|^{\rho I}$, then $w \in \|\mathtt{BELIEF}(v, \alpha)\|^{\rho I}$) and $u \in \mathcal{E}(w, t, \|\phi\|^{\rho I})$ and $y \in \mathcal{E}(w, v, \|\phi\|^{\rho I})$ then (If $w \in \|\mathtt{BELIEF}(u, \beta)\|^{\rho I}$, then $w \in \|\mathtt{BELIEF}(y, \beta)\|^{\rho I}$).

The axioms AE1-AE3 state that if u is the expanded world of t with respect to ϕ then the beliefs at t are preserved at u and the agent acquires belief in ϕ at u. It does not rule out the possibility of the agent acquiring beliefs which are in no way related to ϕ nor the original world at time t. In other words the *only* beliefs at u are the beliefs at t and the belief in ϕ and all its consequences, where ϕ is a first-order sentence with respect to which the expansion is being carried out. This is equivalent to *minimizing the acquisition of beliefs* during expansion. Thus any belief formula at u which is not already in t has been obtained by an explicit expansion or is a consequence of an explicit expansion.

Axiom of Minimization of Beliefs

(AE5) $\mathtt{BELIEF}(u, \phi) \rightarrow \mathtt{BELIEF}(t, \phi) \vee (\mathtt{EXPAND}(t, \alpha, u) \wedge \mathtt{BELIEF}(u, \mathtt{BELIEF}(u, \alpha) \supset \phi))$.

Semantically the above axiom translates to the following condition,

(CE5) If $w \in \|\mathtt{BELIEF}(u, \phi)\|^{\rho I}$ then ($w \in \|\mathtt{BELIEF}(t, \phi)\|^{\rho I}$ or ($u \in \mathcal{E}(w, t, \|\alpha\|^{\rho I})$ and $w \in \|\mathtt{BELIEF}(u, \mathtt{BELIEF}(u, \alpha) \supset \phi)\|^{\rho I}$)).

While axioms (AE2), (AE3) and (AE4) enforce the principle of *maximal coherence*, axioms (AE1) and (AE5) enforce the principle of *minimal change*.

The following inference rule which states that if ϕ_1 and ϕ_2 are equivalent then expanding with respect to either one of them will give the same result is also needed. Modal systems which are closed under this type of inference rule are called *classical* systems [3].

(RE1) From $\vdash \phi_1 \equiv \phi_2$ infer $\vdash \mathtt{EXPAND}(t, \phi_1, u) \equiv \mathtt{EXPAND}(t, \phi_2, u)$.

The axioms and inference rule (AE1)-(AE5) and (RE1) capture the proof-theoretic notion of *minimal change* and *maximal coherence* for expansion. The conditions (CE1)-(CE5), on the expansion function and the belief relation capture the semantic notion of the *closest* possible worlds which obey the principles of minimal change and maximal coherence. The semantic conditions (CE1)-(CE5) describe a *closest expanded world.* The class of models whose $\mathcal{E}$ selection function satisfies the conditions (CE1)-(CE5) are called the $\mathcal{E}$-models.

4.2 Contraction

The axioms of contraction are very similar to the axioms of expansion. The axiom corresponding to the axiom of inclusion for expansion, is the *Axiom of Exclusion* for contraction. Thus if the world t is contracted with respect to ϕ to give u then the agent must not believe ϕ at u.

Axiom of Exclusion

(AC1) $\mathtt{CONTRACT}(t, \phi, u) \rightarrow \neg\mathtt{BELIEF}(u, \phi)$

Semantically the above axiom states that,

(CC1) If $u \in \mathcal{C}(w, t, \|\phi\|^{\rho I})$ then $w \in W - \|\mathtt{BELIEF}(u, \phi)\|^{\rho I}$.

Just as expansion preserves the beliefs of an agent one would expect contraction to preserve the non-beliefs of an agent. In fact, this is the case if one assumes that the world does not contain any AE belief formulas, i.e. formulas of the form $\neg\mathtt{BELIEF}(t, \phi) \supset \psi$. For such cases, the following *Axiom of Preservation of Non-beliefs* which is analogous to the Axiom of Preservation of Beliefs holds.

Axiom of Preservation of Non-Beliefs

$$\mathtt{CONTRACT}(t, \phi, u) \rightarrow (\neg\mathtt{BELIEF}(t, \alpha) \rightarrow \neg\mathtt{BELIEF}(u, \alpha))$$

However, in the more general case, where the world contains both AE and universal AE beliefs the above axiom does not hold. Consider the case where the agent contracts the formula ϕ to go into a time point u, where the AE belief $\neg\mathtt{BELIEF}(u, \phi) \supset \psi$ is true. Now as the agent contracts ϕ, the formula $\neg\mathtt{BELIEF}(u, \phi)$ will be true and this will cause ψ to be true and hence ψ to be believed. Thus in the presence of AE beliefs the agent can acquire new beliefs during contraction. The acquisition of these beliefs should be minimized. Hence the new beliefs acquired by the agent should be the logical consequences of $\neg\mathtt{BELIEF}(u, \phi)$, where ϕ is the formula with respect to which the agent has performed contraction. The following axiom called the *Axiom of Minimization of Beliefs* is analogous to the corresponding axiom of expansion.

Axiom of Minimization of Beliefs

(AC2) $\mathtt{BELIEF}(u, \phi) \rightarrow \mathtt{BELIEF}(t, \phi) \vee (\mathtt{CONTRACT}(t, \alpha, u) \wedge \mathtt{BELIEF}(u, \neg\mathtt{BELIEF}(u, \alpha) \supset \phi))$.

Semantically the above axiom translates to the following condition,

(CC2) If $w \in \|\mathtt{BELIEF}(u, \phi)\|^{DI}$ then $(w \in \|\mathtt{BELIEF}(t, \phi)\|^{DI}$ or $(u \in \mathcal{C}(w, t, \|\alpha\|^{DI})$ and $w \in \|\mathtt{BELIEF}(u, \neg\mathtt{BELIEF}(u, \alpha) \supset \phi)\|^{DI}))$.

If the agent does not believe ϕ at t then contraction with respect to ϕ will not increase the non-beliefs of the agent. In other words under the above conditions the contraction operation preserves the agent's beliefs as well as non-beliefs. This results in the following *Axiom of Trivial Contraction*,

Axiom of Trivial Contraction

(AC3) $\neg\mathtt{BELIEF}(t, \phi) \wedge \mathtt{CONTRACT}(t, \phi, u) \rightarrow (\neg\mathtt{BELIEF}(t, \alpha) \equiv \neg\mathtt{BELIEF}(u, \alpha))$.

The semantic condition for the above axiom is as follows,

(CC3) If $w \in W - \|\mathtt{BELIEF}(t, \phi)\|^{DI}$ and $u \in \mathcal{C}(w, t, \|\phi\|^{DI})$ then $(w \in W - \|\mathtt{BELIEF}(t, \alpha)\|^{DI}$, iff $w \in W - \|\mathtt{BELIEF}(u, \alpha)\|^{DI})$.

The axiom of monotonicity is not satisfied by contraction. This is because contraction reduces the beliefs of the agent and is therefore non-monotonic in nature.

While axiom (AC2) stated what are the formulas which should be acquired in the contracted world, axiom (AC5) states what are the beliefs which should be given up during contraction. It states this indirectly by requiring whatever is *lost* during contraction should be *recovered* during expansion. As the agent gains as little as possible during expansion (due to the Axiom of Minimization of Beliefs), the agent has to loose as little as possible for the following axiom to hold. This implies that the following axiom performs the function of minimizing non-beliefs and is called the *Axiom of Minimization of Non-beliefs*.

Axiom of Minimization of Non-beliefs

(AC4) $\mathtt{CONTRACT}(t, \phi, u) \wedge \mathtt{EXPAND}(u, \phi, v) \rightarrow (\mathtt{BELIEF}(t, \alpha) \rightarrow \mathtt{BELIEF}(v, \alpha))$.

Semantically, the above axiom is equivalent to the following condition,

(CC4) If $u \in \mathcal{C}(w, t, \|\phi\|^{DI})$ and $v \in \mathcal{E}(w, u, \|\phi\|^{DI})$ then $(w \in \|\mathtt{BELIEF}(t, \alpha)\|^{DI}$, then $w \in \|\mathtt{BELIEF}(v, \alpha)\|^{DI})$.

A strict equivalence of the consequent of (AC4) does not hold. Consider the case where ϕ is not present in t. Then according to axiom (AC3) the time points t and u will have identical beliefs. Now if u is expanded with respect to ϕ, then at v the agent will believe in ϕ but did not have this belief at t, which proves that the strict equivalence does not hold.

The following inference rule which states that if ϕ_1 and ϕ_2 are equivalent then contracting with respect to either one of them will give the same result is also needed.

(RC1) From $\vdash \phi_1 \equiv \phi_2$ infer $\vdash \mathtt{CONTRACT}(t, \phi_1, u) \equiv \mathtt{CONTRACT}(t, \phi_2, u)$.

Axioms (AC1)-(AC4) together capture the proof-theoretic notion of *maximal coherence* and *minimal change* for contraction. The conditions (CC1)-(CC4), on the contraction function and the belief relation capture the semantic notion of the *closest* possible worlds which obey the principles of minimal change and maximal coherence. The semantic conditions (CC1)-(CC4) describe a *closest contracted world*. The class of models whose $\mathcal{C}$ selection function satisfies the conditions (CC1)-(CC4) are called the $\mathcal{C}$-models.

4.3 Revision

As revision can be expressed in terms of expansion and contraction only one axiom is needed for revision. This axiom states that revising with respect to ϕ is equivalent to contracting with respect to $\neg\phi$ followed by expansion with respect to ϕ. This axiom is a reformulation of Levi identity [6].

Axiom of Revision

(AR1) $\mathtt{REVISE}(t, \phi, u) \rightarrow \mathtt{CONTRACT}(t, \neg\phi, v) \wedge \mathtt{EXPAND}(v, \phi, u)$

The axiom system for the language L_2, called the *basic dynamic (BD)-model*, is the B-modal system together with the axioms and inference rules for expansion, contraction and revision. The *basic* $\mathcal{D}$-model is defined as a model of the dynamic interpretation DI, whose $\mathcal{B}$ relation is a $\mathcal{B}$-model, whose $\mathcal{E}$ function is a

$\mathcal{E}$-model and whose $\mathcal{C}$ function is a $\mathcal{C}$-model.

The soundness and completeness of the dynamic reasoning system is stated below. Once again the details of the proof can be found in [17].

Theorem 4.2: The BD-modal system is sound and complete with respect to the class of all basic $\mathcal{D}$-models.

4.4 Actions and Planning

Expansion, contraction and revision are the most primitive or fundamental dynamic operations. However, they are not the only dynamic operations. The dynamic operation which has received a great deal of attention in AI is the notion of *actions*. In the situation calculus [13] approach actions are treated as transformations from one situation to another. Situations are like possible worlds, introduced in the previous section. Actions can be defined in terms of the dynamic operations expansion, contraction and revision. The dynamic operations can also be used to define parallel actions. This helps in providing a unifying architecture for dynamic reasoning as well as planning. This section gives a brief description of how to define actions in terms of the dynamic operations and provides insights into a completely integrated autonomous reasoning and planning system.

Actions are normally defined in terms of *preconditions* and *postconditions*. If a certain precondition holds at the current instant of time and an action is carried out then the post-condition holds in the next time instant. This can be expressed as the modal formula ACTION(t, β, α, γ, u), which states that at t if the agent believes β and the action α is carried out then the agent will believe in γ at the next time instant u. The axiom for action is defined as follows,

Axiom of Action

ACTION(t, β, α, γ, u) ≡ BELIEF(t, β) ∧ REVISE(t, γ, u).

The same methodology as above can be used to define both *sequential* and *parallel* actions. Let α_1 ; α_2 denote two sequential actions and $\alpha_1 \parallel \alpha_2$ denote two parallel actions. The axioms for these actions can be defined as follows,

Axiom of Sequential Action

ACTION(t, $\beta_1 \wedge \beta_2$, α_1 ; α_2, $\gamma_1 \wedge \gamma_2$, u) ≡ BELIEF(t, β_1) ∧ REVISE(t, γ_1, v) ∧ BELIEF(v, β_2) ∧ REVISE(v, γ_2, u).

Axiom of Parallel Action

ACTION(t, $\beta_1 \wedge \beta_2$, $\alpha_1 \parallel \alpha_2$, $\gamma_1 \wedge \gamma_2$, u) ≡ BELIEF(t, $\beta_1 \wedge \beta_2$) ∧ REVISE(t, γ_1, u) ∧ REVISE(t, γ_2, u).

The following example provides a simple situation, where the theory developed in this paper can be used.

Example:

Consider the blocks-world, at time t1, where there are two blocks A and B, such that the Red colored block, A, is at location L1 and the Blue colored block, B, is at location L2. Also both blocks A and B are clear. The laws in this domain can be stated as follows, a) No two blocks are on the same location, b) No block occupies more than one location, c) A clear location has nothing on top of it, and d) No block has more than one color. This information is stated as follows in the dynamic belief system.

1.1 ∀t(BELIEF(t, on(x,l) ∧ x ≠ y → ¬on(y,l)))
1.2 ∀t(BELIEF(t, on(x,l) ∧ l ≠ m → ¬on(x,m)))
1.3 ∀t(BELIEF(t, clear(l) ≡ ¬on(x,l)))
1.4 ∀t(BELIEF(t, color(x,c) ∧ c ≠ d → ¬color(x,d)))

The contingent information about the blocks world at time t1 is stated as follows,

1.5 BELIEF(t1, clear(A)) 1.6 ¬BELIEF(t1, ¬clear(A))
1.7 BELIEF(t1, clear(B)) 1.8 ¬BELIEF(t1, ¬clear(B))
1.9 BELIEF(t1, on(A,L1)) 1.10 ¬BELIEF(t1, ¬on(A,L1))
1.11 BELIEF(t1, on(B,L2)) 1.12 ¬BELIEF(t1, ¬on(B,L2))
1.13 BELIEF(t1, color(A,Red)) 1.14 ¬BELIEF(t1, ¬color(A,Red))
1.15 BELIEF(t1, color(B,Blue)) 1.16 ¬BELIEF(t1, ¬color(B,Blue))

Using the laws 1.1 to 1.4 and the beliefs 1.5 to 1.16 the following additional beliefs can be derived,

1.17 BELIEF(t1, ¬on(A,B)) 1.18 ¬BELIEF(t1, on(A,B))
1.19 BELIEF(t1, ¬on(B,A)) 1.20 ¬BELIEF(t1, on(B,A))

1.21 `BELIEF(t1, ¬on(A,L2))` 1.22 `¬BELIEF(t1, on(A,L2))`
1.23 `BELIEF(t1, ¬on(B,L1))` 1.24 `¬BELIEF(t1, on(B,L1))`
1.25 `BELIEF(t1, ¬color(A,Blue))` 1.26 `¬BELIEF(t1, color(A,Blue))`
1.27 `BELIEF(t1, ¬color(B,Red))` 1.28 `¬BELIEF(t1, color(B,Red))`

Assume that there are two robots: the painting robot and the block moving robot. Let the painting robot paint the block `B` using `Red` paint and at the same time let the block-moving robot move the block `B` to location `A`. The action `paint(B,Red)` has no preconditions but it causes the color of `B` to be `Red`. The action `move(B,A)` requires that both the blocks `B` and `A` be clear. The post-condition of performing the action `move(B,A)` is that the block `B` is on `A`. As the preconditions of both the actions are believed at `t1`, both the actions can be carried out in parallel. From the axiom of parallel actions this is equivalent to revising `t1` with respect to `on(B, A)` and with respect to `color(B, Red)`, to give the time point `t2`. Using the axioms of expansion, contraction and revision the time point `t2` can be *derived* from the time point `t1`. The state of the world at `t2` is given below.

2.5 `BELIEF(t2, ¬clear(A))` 2.6 `¬BELIEF(t2, clear(A))`
2.7 `BELIEF(t2, clear(B))` 2.8 `¬BELIEF(t2, ¬clear(B))`
2.9 `BELIEF(t2, on(A,L1))` 2.10 `¬BELIEF(t2, ¬on(A,L1))`
2.11 `BELIEF(t2, ¬on(B,L2))` 2.12 `¬BELIEF(t2, on(B,L2))`
2.13 `BELIEF(t2, color(A,Red))` 2.14 `¬BELIEF(t2, ¬color(A,Red))`
2.15 `BELIEF(t2, ¬color(B,Blue))` 2.16 `¬BELIEF(t2, color(B,Blue))`
2.17 `BELIEF(t2, ¬on(A,B))` 2.18 `¬BELIEF(t2, on(A,B))`
2.19 `BELIEF(t2, on(B,A))` 2.20 `¬BELIEF(t2, ¬on(B,A))`
2.21 `BELIEF(t2, ¬on(A,L2))` 2.22 `¬BELIEF(t2, on(A,L2))`
2.23 `BELIEF(t2, ¬on(B,L1))` 2.24 `¬BELIEF(t2, on(B,L1))`
2.25 `BELIEF(t2, ¬color(A,Blue))` 2.26 `¬BELIEF(t2, color(A,Blue))`
2.27 `BELIEF(t2, color(B,Red))` 2.28 `¬BELIEF(t2, ¬color(B,Red))`

Note that not only the formula `on(B,A)` is believed, but all its consequences, namely `¬clear(A)` and `¬on(B,L2)` are also believed. Similarly the formula `color(B,Red)` and its consequence `¬color(B,Blue)` are also believed. Thus the principles of minimal change and maximal coherence solves two of the important problems in planning; the frame problem [13] and ramification problem [9]. An alternative approach to planning using the principles of minimal change and maximal coherence at the meta-level is discussed in [5].

In order to obtain a completely autonomous reasoning system, capable of dynamic reasoning and reactive planning, the architecture mentioned in this paper should be extended to a belief-desire-intention architecture [16]. The principles of minimal change and maximal coherence can then be used not only for change in beliefs, but also for change in desires and intentions. Such a system would act like a dynamic reasoning system as described in this paper and as a reactive planning system as described in [8]. Such a system would be able to react rationally, within a reasonable amount of time, without any human intervention, to any unforeseen situation and will be ideal for space-based applications.

5. Conclusion

This paper presents a dynamic belief system capable of representing and reasoning about change. The dynamic reasoning system based on the dynamic belief system embodies two general principles of change, namely minimal change and maximal coherence. The dynamic reasoning system can be used as a module for a more general purpose autonomous reasoning system capable of reacting to any unforeseen circumstances. Such systems will prove to be crucial for future space based AI systems.

Acknowledgements
The research was partially supported by the Sydney University Postgraduate Research Award. The research was carried out when the authors were visiting IBM. The authors wish to express their thanks to IBM for its generous funding during this period.

References

1. Alchourron, C., Gardenfors, P., and Makinson, D., "On the Logic of Theory Change: Partial Meet Contraction Functions and their associated Revision Functions," *Journal of Symbolic Logic* **50**, pp. 510-530 (1985).
2. Chellas, B. F., "Basic Conditional Logic," *Journal of Philosophical Logic* **4**, pp. 133-153 (1975).
3. Chellas, B. F., *Modal Logic: An Introduction*, Cambridge University Press (1980).
4. Doyle, J., "A Truth Maintenance System," *Artificial Intelligence* **12**, pp. 231-272 (1979).
5. Foo, N.Y. and A.S.Rao, "Belief Revision in a Microworld," Technical Reort No. 325 , Department of Computer Science, University of Sydney, Sydney (1988).
6. Gardenfors, P., *Knowledge in Flux: Modeling the Dynamics of Epistemic States*, Bradford Book, MIT Press, Cambridge, Mass. (1988).
7. Georgeff, M. P. and Lansky, A. L., "Procedural Knowledge," *Proceedings of the IEE Special Issue on Knowledge Representation* **74**, pp. 1383-1398 (1986).
8. Georgeff, M. P., Lansky, A. L., and Schoppers, M. J., "Reasoning and Planning in Dynamic Domains: An Experiment with a Mobile Robot," Technical Note 380, SRI International, Menlo Park, California (1987).
9. Ginsberg, M. L. and Smith, D. E., "Reasoning About Action I: A Possible Worlds Approach," in *The Frame Problem in Artificial Intelligence*, ed. F. M. Brown, Morgan Kaufmann, Los Altos (1987).
10. Halpern, J. Y. and Moses, Y. O., "A Guide to the Modal Logics of Knowledge and Belief," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence(IJCAI-85)*, Los Angeles (1985).
11. Kleer, J. de, "An Assumption-based TMS," *Artificial Intelligence* **28**, pp. 127-162 (1986).
12. Lewis, D., *Counterfactuals*, Harvard University Press, Cambridge, Mass. (1973).
13. McCarthy, J. and Hayes, P. J., "Some Philosophical Problems from the Standpoint of Artificial Intelligence," pp. 463-502 in *Machine Intelligence*, ed. D. Michie, American elsevier, New York (1969).
14. Mendelson, E., *Introduction to Mathematical Logic*, D. Van Nostrand Company, New York (1979).
15. Moore, R. C., "Semantical Considerations on Nonmonotonic Logic," *Artificial Intelligence* **25** (1985).
16. Pollack, M. E., Israel, D. J., and Bratman, M. E., "Towards an Architecture for Resource-bounded Agents," Technical Note 425 , SRI International, Menlo Park (1987).
17. Rao, A. S., "Dynamics of Belief Systems: A Philosophical, Logical and AI Perspective," PhD Thesis(in preparation), , Department of Computer Science, University of Sydney, Sydney (1988).

N89-15587

Strategies for Adding Adaptive Learning Mechanisms to Rule-Based Diagnostic Expert Systems*

D.C. St. Clair**,
C. L. Sabharwal
University of MO--Rolla
Graduate Engineering Center
St. Louis, MO 63121

W.E. Bond,
Keith Hacke
McDonnell Douglas Research Laboratories
St. Louis, MO 63166

ABSTRACT

Rule-based diagnostic expert systems can be used to perform many of the diagnostic chores necessary in today's complex space systems. These expert systems typically take a set of symptoms as input and produce diagnostic advice as output. The primary objective of such expert systems is to provide accurate and comprehensive advice which can be used to help return the space system in question to nominal operation.

The development and maintenance of diagnostic expert systems is time and labor intensive since the services of both knowledge engineer(s) and domain expert(s) are required. The use of adaptive learning mechanisms to incrementally evaluate and refine rules promises to reduce both time and labor costs associated with such systems. This paper describes the basic adaptive learning mechanisms of strengthening, weakening, generalization, discrimination, and discovery. Next, basic strategies are discussed for adding these learning mechanisms to rule-based diagnostic expert systems. These strategies support the incremental evaluation and refinement of rules in the knowledge base by comparing the set of advice given by the expert system (A) with the correct diagnosis (C). Techniques are described for selecting those rules in the knowledge base which should participate in adaptive learning.

The strategies presented may be used with a wide variety of learning algorithms. Further, these strategies are applicable to a large number of rule-based diagnostic expert systems. They may be used to provide either immediate or deferred updating of the knowledge base.

INTRODUCTION

The basic architecture of rule-based diagnostic expert systems is shown in Figure 1. Symptoms describing the failure to be diagnosed are entered into the expert system via the user interface. The system then "reasons" over the set of symptoms, asking for additional information if necessary. At the conclusion of the "reasoning" process, the expert system provides suggestions for correcting the anomalies of the system under diagnosis. As can be seen in

---

* This work was supported by the McDonnell Douglas Independent Research and Development program and by Statement of Work WS-MDRL-2703 with the University of MO-Rolla Graduate Engineering Center.

** D. St. Clair's Bitnet address is C0567@UMRVMB.

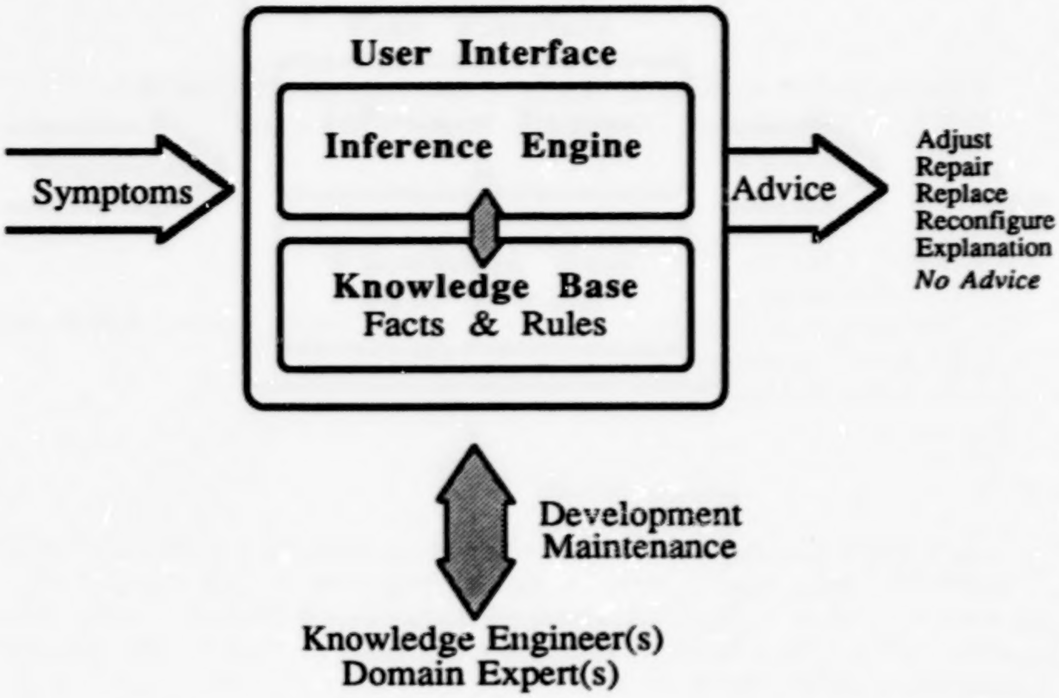


Figure 1. Rule-Based Diagnostic Expert System Architecture.

Figure 1, this advice can take one or more forms. It is also possible that the system will be unable to provide any advice. Of course, the objective of a well-designed expert system is to provide both accurate and comprehensive advice.

The basic components of a rule-based expert system are a knowledge base and an inference engine. The knowledge base consists of a set of facts and a set of rules. Although actual knowledge representations vary from application to application, the rules are logically equivalent to the form:

$$\mathbf{H} \rightarrow \mathbf{K}$$

where set **H** contains one or more elements from the description space. The description space, determined by knowledge engineers and domain experts, contains all the propositions and negated propositions used to describe the environment. Set **K** contains a set of actions to be performed. These actions may modify the knowledge base and/or cause external actions to be performed. Without loss of generality, the remaining discussion assumes that each rule conclusion contains a single action such as "replace part C6."

Clearly, expert system performance is directly related to the accuracy and comprehensiveness of rules in the knowledge base. One basic approach to evaluating rule accuracy and comprehensiveness has been to compare the system's advice with the correct diagnosis. Correct diagnoses come from domain experts themselves and/or from observing the actions required to remove anomalies from the system being repaired. Once this comparison is made, a form of learning is initiated in an effort to upgrade the knowledge base and thus improve the expert system's diagnostic accuracy. The most common approach to learning in this situation utilizes domain experts and knowledge engineers to manually revise the contents of the knowledge base. Although St. Clair et al. [10] have suggested a technique to assist in this endeavor, manual updating of rules is still time and labor intensive.

Attempts are currently under way to develop machine learning techniques which can be used to automate revision of the knowledge base. Both nonincremental and incremental techniques have been developed. Nonincremental learning techniques perform all learning at a given point in time. While the learning process may be repeated periodically, these techniques require that all accumulated test cases be processed at once [4,7]. Other techniques use incremental learning mechanisms which continuously update portions of the knowledge base [9,11]. They attempt to improve the accuracy of advice by refining the components of existing rules. Incremental learning techniques seek to improve the comprehensiveness of advice by creating new rules as necessary.

The following section outlines some of the basic adaptive learning mechanisms. It is followed by a discussion of how expert system advice is classified. Then, strategies for adding adaptive learning mechanisms to rule-based diagnostic expert systems are discussed.

BASIC ADAPTIVE LEARNING MECHANISMS

In diagnostic expert systems, adaptive learning mechanisms utilize the input symptoms, the current knowledge base, and the correct diagnosis to produce an updated knowledge base. Approaches range from simple to highly specialized. Figure 2 illustrates the basic relationship of these components. The adaptive learner continuously applies various learning mechanisms to selected rules in the knowledge base.

Adaptive learning strategies fall into one of six basic categories [1,3,9]: strengthening, weakening, unlearning, generalization, discrimination, and discovery. Strengthening and weakening mechanisms are used to reward correct rules and to penalize incorrect rules. They

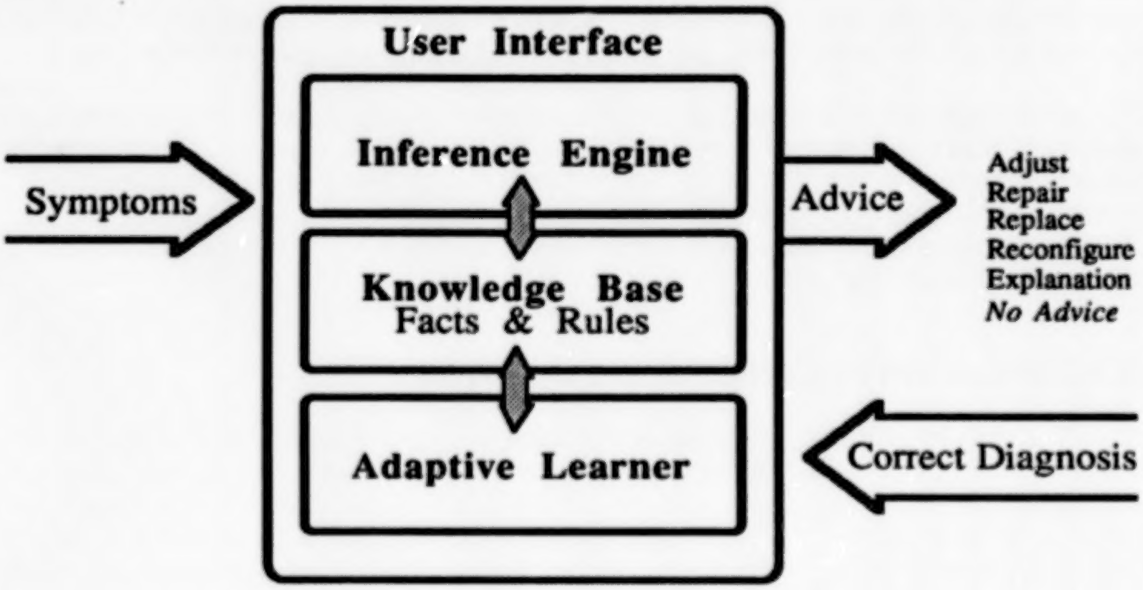


Figure 2. Adaptive Rule-Based Diagnostic Expert System Architecture.

may be implemented by keeping individual rule statistics about the number of times the rule has fired and the number of times it participated in a rule chain leading to an element of the correct diagnosis. These experience indicators can be used to effect future rule firings and to indicate the strength of the advice given in the expert system output.

Unlearning has received little attention in the literature [5]. Unlearning involves the removal of undesirable rules from the knowledge base. Experience indicators may be useful in deciding which rules to remove.

Generalization is the process of reducing the number of propositions and/or negated propositions in a rule's hypotheses. The overall result is to make the hypotheses less restrictive. The rule then becomes applicable in a larger number of situations. The need for generalization is easily identifiable in situations where a rule which should have fired did not fire. This condition, called an error of omission, usually indicates that the rule hypotheses were overly restrictive. Generalizing a set of rules may make it possible to combine several rules into one. Good generalization mechanisms are hard to define due to the difficulty of deciding which propositions can be removed from the rule hypotheses. Bundy et al. [3] provide some suggestions along this line.

Discrimination produces results which are essentially the opposite of generalization. In discrimination, propositions or negated propositions are added to a rule's hypotheses to restrict its firing. Discrimination mechanisms are usually easier to define than generalization mechanisms. Some algorithms [7,9] treat discrimination and generalization as complementary processes. Bundy et al. [3] gives an example to show that they are not fully complementary.

Discovery mechanisms utilize input symptoms and the current knowledge base to create new rules. These mechanisms are necessary whenever the expert system gives incomplete diagnostic advice or no advice at all. They must decide which propositions and negated propositions from the description space should constitute the hypotheses of each new rule. The list of symptoms is generally quite helpful in this regard. Discovery mechanisms may become quite complex in situations where extensive new rule chains must be constructed. As is indicated in a later section, the most difficult part of discovery learning is deciding when to apply it.

Both heuristic and analytical approaches are being applied in an effort to develop a good general set of adaptive learning mechanisms. The analytical approaches include the incremental and nonincremental classes of techniques mentioned earlier. Many approaches attempt to perform logical equivalents of the basic mechanisms described. In cases where knowledge relationships are not complex, a set of simple heuristics may suffice for implementing the learning mechanisms [9].

CLASSIFICATION OF EXPERT SYSTEM ADVICE

The foundation for deciding how to add adaptive learning mechanisms to rule-based diagnostic expert systems is based on comparing expert system advice with correct diagnoses. Accordingly, let **A** denote the advice set produced by a diagnostic expert system in response to a given set of input symptoms. The elements of **A** are the consequents produced as a result of one or more rule chains fired by the inference engine. As indicated earlier, assume that each rule chain terminates with a consequent containing a single piece of diagnostic advice such as "adjust

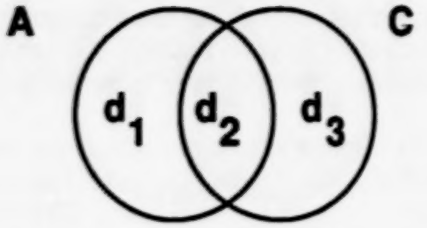


Figure 3. Comparison of Expert System Advice with Correct Diagnosis.

part D5." Further, let **C** represent the correct diagnosis. As suggested by St. Clair et al. [10], the comparison of these two sets forms the foundation for evaluating expert system output. Figure 3 illustrates the three cases which may arise.

The elements of set **A** represent components of diagnostic expert system advice while the elements of set **C** represent components of the correct diagnosis. Hence, $d_2 \in \mathbf{A} \cap \mathbf{C}$, shown in Figure 3, represents a correct piece of advice which was given by the expert system. The associated rules have produced a correct system response. The element $d_1 \in \mathbf{A} - \mathbf{C}$ represents an advice component which is incorrect. The associated rules have produced an incorrect system response and need revision. The element $d_3 \in \mathbf{C} - \mathbf{A}$ represents a case in which a component of the correct diagnosis was not included as part of the expert system's advice. This occurrence indicates a condition where the expert system has failed to provide needed advice. Since two different rule chains may produce the same advice, the components of these sets may not be distinct.

Note that the conditions illustrated by Figure 3 are a comparison of expert system output with known correct diagnosis. Such a comparison will not identify cases in which incorrect rule chains produce correct conclusions. In addition, some erroneous conditions, such as conflicting conclusions, cannot be completely uncovered by comparing the contents of sets **A** and **C**.

STRATEGIES FOR ADDING ADAPTIVE LEARNING MECHANISMS

Given a rule-based diagnostic expert system, a set of input symptoms, and the correct diagnosis for this set of symptoms, the adaptive learner must decide how and when to apply the various types of adaptive learning mechanisms described earlier (see Figure 2). These mechanisms may modify rule statistics, modify rule hypotheses, and/or create new rules. The choice of which strategies to apply is based on comparing expert system advice with the correct diagnosis as discussed above and illustrated in Figure 3.

Rule chains terminating in $d_i \in \mathbf{A} - \mathbf{C}$ represent cases in which the expert system gave incorrect advice. At least one rule in each chain has committed an error of commission by firing when it should not have fired. The strategy for adding adaptive learning mechanisms calls for discrimination to be performed to restrict the firing of the rule. In one prototype system [9], the discrimination algorithm was implemented by simply replacing the existing rule by a rule whose hypotheses were chosen from the set of input symptoms and whose conclusion was d_i. Such a simple strategy may not work well in situations in which the rule being modified participates in several other rule chains. Rules participating in more than one rule chain will have experience indicators which vary from the experience indicators of other rules in the chain.

Advice components $d_i \in \mathbf{A} \cap \mathbf{C}$ represent cases in which the expert system gave correct advice. Two situations need to be noted in this case. The first situation involves updating both experience indicators for all rules participating in the rule chain. This action does not indicate that each rule in the chain is correct but only that it has participated in a rule chain leading to a correct conclusion. The second situation calls for deciding whether or not generalization should be performed. For instance, if two or more $d_i \in \mathbf{A} \cap \mathbf{C}$ have the same conclusion, generalization may be desirable. The decision of whether or not to generalize is difficult, as the following three simple rules demonstrate.

Rule 1:
If part number = B10
measured value = 1011,
temperature range = (70 79)
intermittent = no
Then
replace part B6.

Rule 2:
If part number = B10
measured value = 1011,
Then
replace part B6.

Rule 3:
If part number = B10
temperature range = (70 79)
Then
replace part B6.

All three rules will fire whenever Rule 1 fires. The question arises as to whether generalization should be used to replace these rules by a more general rule, and if so, what should be contained in the hypotheses of the new rule. This is a difficult question at best. The experience indicators utilized in conjunction with some of the techniques mentioned in the previous section may help resolve such issues. Even though Rule 1 is the least general, if its accuracy rate is high and its times fired statistic is close to that of the other rules, it may be the case that it should be retained and the remaining rules should be removed from the knowledge base.

Those $d_i \in \mathbf{C} - \mathbf{A}$ indicate cases in which the expert system's knowledge base needs revision. If one or more rules committed an error of omission by failing to fire when they should, the knowledge base is inaccurate. On the other hand, if the knowledge base does not contain information pertinent to the diagnostic action d_i, it is incomplete. The difficulty in deciding which case applies is related directly to the complexity of the knowledge base and to the rule chains it produces.

An error of omission occurs if the knowledge base contains one or more unexecuted rule chains which terminate in the appropriate diagnostic action $d_i \in \mathbf{C} - \mathbf{A}$. These rule chains may not have been executed because one or more hypotheses in the rule chain were not satisfied. In the example rules stated above, assume that of the three rules given, Rule 1 is currently the only one in the knowledge base and that it did not fire because the expert system had no knowledge of the current temperature range attribute. In addition, assume that the correct diagnostic advice was to "replace part B6." If the missing attribute is not important to the performance of the rule, the error of omission can be corrected by generalizing the rule and removing the temperature range attribute. Depending on the complexity of the knowledge base and the rule chains generated, finding the rules to generalize and performing the generalization may be a complex process.

The knowledge base is incomplete when a rule cannot be found which is a candidate for generalization. In this case, the knowledge base does not contain information pertinent to the diagnostic action $d_i \in \mathbf{C} - \mathbf{A}$. In this situation, a discovery learning mechanism should be invoked to capture the missing knowledge. Depending on the structure of the knowledge base, the discovery learning mechanism may be either simple or complex. One simple discovery algorithm uses the value of the input symptoms as the hypotheses of the new rule and the correct diagnostic action as the rule consequent [9].

The first step in improving existing rules is to identify which rules should be revised. This necessitates recording the trace of each rule chain producing system output along with corresponding rule unifications. Whenever the adaptive learner is invoked, one or both of the rule statistics must be updated. If the rule has participated in a rule chain leading to a component of set **C**, both of the rule's experience indicators should be incremented. If the rule has participated in a rule chain leading to a component of set **A** - **C**, only the times fired statistic should be incremented. The trace provides quick access to these statistics.

In cases where a rule chain terminates with an element $d_i \in \mathbf{A} - \mathbf{C}$, the faulty rules must be identified. Bundy et al.[3] describe two basic techniques utilized by rule learning programs to identify faulty rules. Both approaches are similar in that they only identify the first faulty rule within a chain.

In the first approach, the actual rule chain is compared with the chain which should have fired. Some programs require this ideal chain as input [2] while others [6] attempt to derive it by analysis using problem-solving and inference techniques. The first difference between the chains indicates the faulty rule. The necessity of identifying the ideal rule chain makes this technique difficult to apply.

The second technique for finding a faulty rule is called Contradiction Backtracking. This technique, developed by Shapiro [8] does not require identification of an ideal chain. Assuming the actual rule chain concludes with d_i, Shapiro's algorithm begins by examining the last resolution step leading to d_i. If the propositions which were resolved to produce d_i are true, select the branch of the tree containing these propositions as part of the rule hypothesis, else select the other branch. Backtracking up the resolution tree continues in this manner until a rule from the rule base is reached. This is the faulty rule. Both Shapiro and Bundy et al. give examples of Contradiction Backtracking.

Unlearning strategies generally require an approach different from those described since deciding which rules to "unlearn" can not generally be determined by comparing the results of sets **A** and **C**. Unlearning strategies require that the experience indicators of each rule in the rule set be evaluated periodically. However, care must be taken not to remove a rule simply because of poor performance. While a rule that is correct six out of a thousand times is not contributing to the overall quality of the knowledge base, it may need to be modified and not merely discarded. In addition, rules having a small number of firings but a relatively high percentage of correct firings probably should be left in the knowledge base. An individual rule's high percentage of correct firings indicates it is participating in correct rule chains. An individual rule's low number of firings may indicate that the conditions it identifies are exceptional cases

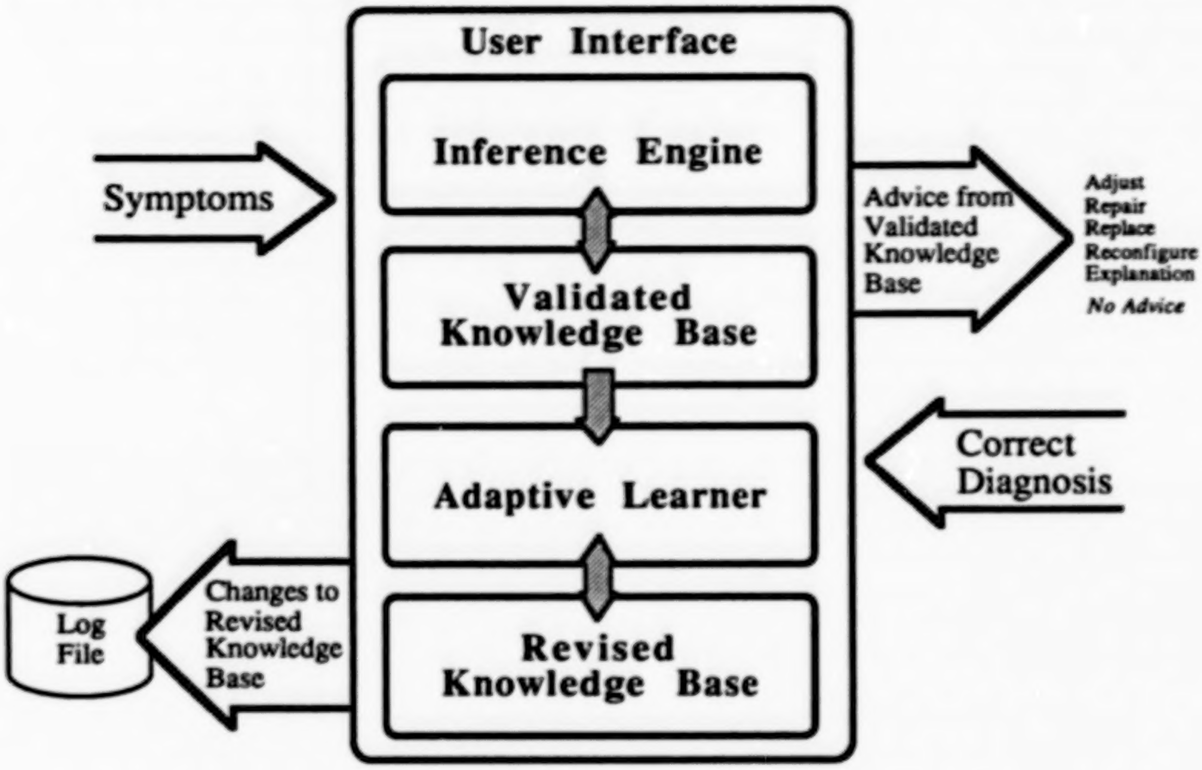


Figure 4. Dual Expert System Architecture.

and not merely noise in the input symptoms. If the times fired and times correct statistics are extremely small, the rule may have been created as the result of noisy input data.

The strategies described provide a means of incrementally updating the knowledge base each time a new piece of information is available. In diagnostic applications where human expert intervention is desired before changes are made to the knowledge base, it may be desirable to use a dual expert system as illustrated in Figure 4. The inference engine first reasons against the symptoms by using the validated knowledge base. It then outputs diagnostic advice. Next, the inference engine repeats the diagnosis using the revised knowledge base to produce a second set of advice. This advice need not be reported to the user. Upon receipt of the correct diagnosis, the adaptive learner updates the revised knowledge base. Knowledge base changes are logged so that they may be reviewed by domain experts. This scenario guarantees that the system always utilizes the validated knowledge base. However, at any point in time, human experts may review the revised knowledge base and, if desirable, use it to replace the validated knowledge base.

CONCLUSIONS

Techniques presented in this paper outline basic adaptive learning mechanisms and strategies for incorporating them into diagnostic expert systems. Although implementations of these strategies and learning mechanisms vary from system to system, the basic concepts are applicable to a large number of diagnostic expert systems. Continued research in these areas promises to reduce the maintenance costs of diagnostic expert systems.

REFERENCES

1. Blaxton T. A. and Kushner, B. G., *An Organizational Framework for Comparing Adaptive Artificial Intelligence Systems*, **1986 Proceedings of the Fall Joint Computer Conference,** IEEE Computer Society, November 1986, pp. 190-199.

2. Brazdil, P., *A Model for Error Detection and Correction*, Ph.D. Dissertation, University of Edinburgh, 1981.

3. Bundy, A., Silver, B., and Plummer, D., *An Analytical Comparison of Some Rule-Learning Programs*, **Artificial Intelligence**, Vol. 27, 1985, pp. 137-181.

4. Cheeseman, P., Kelly, J., Self, M., Stutz, J., Taylor, W., and Freeman, D., *AutoClass: A Bayesian Classification System*, **Proceedings of the Fifth International Conference on Machine Learning**, Morgan Kaufmann Publishers, Inc., June 1988, pp. 54-64.

5. Markovitch, S. and Scott, P. D., *The Role of Forgetting in Learning*, **Proceedings of the Fifth International Conference on Machine Learning**, Morgan Kaufmann Publishers, Inc., June 1988, pp. 459-465.

6. Mitchell, T.M., Utgoff, P.E., and Banerji, R., *Learning by Experimentation: Acquiring and Modifying Problem-Solving Heuristics*, **Machine Learning**, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Ed.s), Palo Alto, CA: Tioga, 1983, pp. 163-190.

7. Quinlan, J.R., *Induction of Decision Trees*, **Machine Learning**, Vol. 1, 1986, pp.81-106.

8. Shapiro, E., *An Algorithm That Infers Theories From Facts*, **Proceedings of the Seventh International Joint Conference on Artificial Intelligence**, Los Altos, CA: William Kaufmann, Inc., 1981, pp. 446-451.

9. St. Clair, D. C., Bond, W. E., Flachsbart, B. B., and Vigland, A. R., *An Architecture for Adaptive Learning in Rule-Based Diagnostic Expert Systems*, **1987 Proceedings of the Fall Joint Computer Conference**, IEEE Computer Society, October 1987, pp. 678-685.

10. St. Clair, D. C., Bond, W. E., and Flachsbart, B. B., *Using Output to Evaluate and Refine Rules in Rule-Based Expert Systems*, **Proceedings of the Third Conference on Artificial Intelligence for Space Applications, Part I,"** NASA Conference Publication 2492, November 1987, pp. 9-14.

11. Utgoff, P. E., *ID5: An Incremental ID3*, **Proceedings of the Fifth International Conference on Machine Learning**, Morgan Kaufmann Publishers, Inc., June 1988, pp. 107-120.

N89-15588

An Architecture for an Autonomous Learning Robot

Brian Tillotson
Boeing Electronics
P.O. Box 24969 M/S 7J-24
Seattle WA 98124-6269

ABSTRACT

An autonomous learning device must solve the *example bounding* problem, i.e., it must divide the continuous universe into discrete examples from which to learn. We describe an architecture which incorporates an example bounder for learning. The architecture is implemented in the GPAL program. An example run with a real mobile robot shows that the program learns and uses new causal, qualitative, and quantitative relationships.

INTRODUCTION

A long term goal of AI research is to produce an autonomous machine that acts as a surrogate for human beings. The machine must perceive the physical environment and be able to change it in useful ways. The machine should accept new goals from a human at any time. When the machine does not know how to achieve its goals or has none pending, it should do experiments to fill gaps in its knowledge. This need for self-guided learning is a major obstacle in building a human surrogate machine.

Machine learning research has rarely addressed the requirement we call *example bounding*. In most machine learning work, the events or examples from which the system learns are somehow bounded or defined before they are input to the learning system. This approach is inadequate for an autonomous machine, which cannot generalize unless it somehow divides the continuous universe into discrete examples. Dieterrich and Michalski[3] briefly address the problem in their program SPARC/E, which plays a card game in which a pattern must be learned. The program is initially unsure whether the pattern is based on the last one, two, or three cards played, so it adjusts the temporal boundaries of events until a meaningful pattern can be recognized.

For humans, example bounding often seems to be guided by temporal or spatial proximity.[1] We have designed and tested an example bounder which uses temporal information to group simple events into examples for inductive learning.[5] If a temporal rule is known to link events of two concepts, then that rule is used to guide the quick, accurate formation of examples. If no such rule is known, then examples are formed by temporal clustering.

Here we propose an autonomous learning device architecture which incorporates an example bounder. We describe the overall architecture and examine each of its major components. We briefly describe an implementation of the architecture, and present an example run. Finally, we discuss the significance of this work and some areas for future work. A glossary is appended at the end of the paper.

OVERVIEW OF PROPOSED ARCHITECTURE

The architecture we propose is illustrated in Figure 1. It has four major components: a planner, a temporal database of events, a body of conceptual knowledge, and a learner. The

PRECEDING PAGE BLANK NOT FILMED

planner controls the creation of new events. Primitive events correspond to actuator commands and sensor inputs; derived events are recognized or computed from simple events or other derived events. The event database maintains information about temporal relationships between events. Conceptual knowledge is divided two ways: a concept is either a descriptor or a rule, and either an hypothesis or a theory. The learner incrementally forms new hypotheses by constructive induction, and uses the planner to carry out experiments which verify or refute hypotheses.

The top-level control of the architecture recognizes three situations. First, if there is a pending goal from the user and the system knows how to achieve the goal, then the planner acts to achieve the goal. Second, for a pending goal which the system does not know how to achieve, the learner concentrates its research on concepts related to the goal. Third, with no pending goals, the learner studies concepts which it heuristically determines to be interesting.

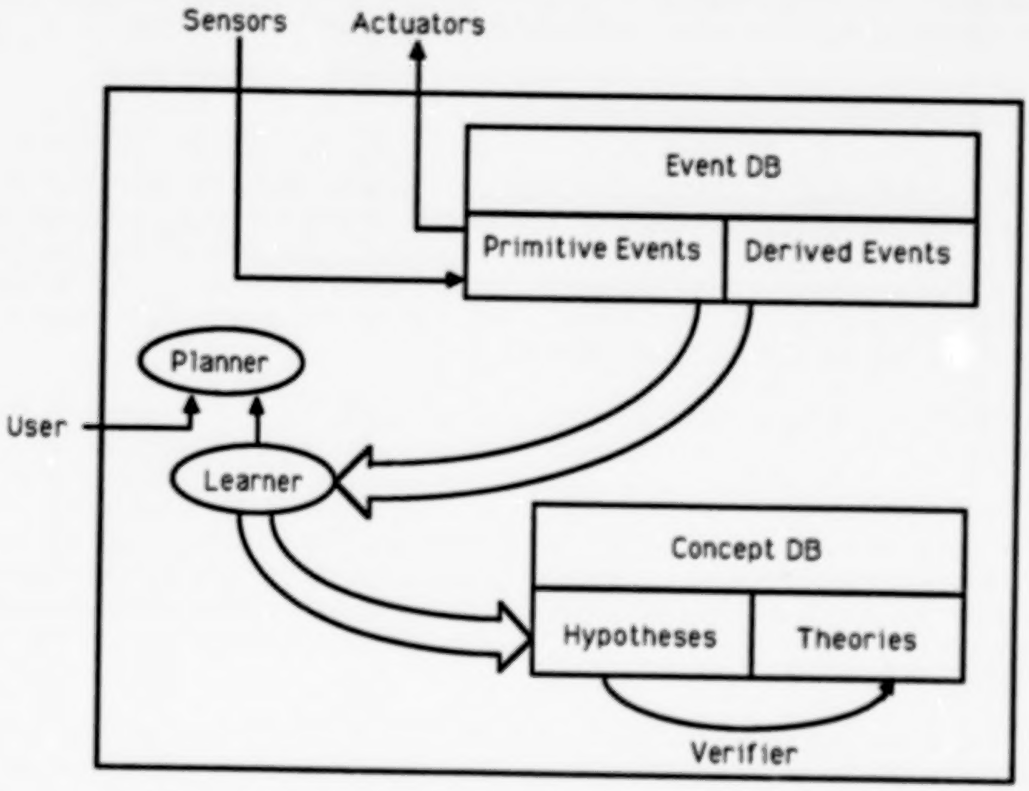


Figure 1. Proposed architecture for autonomous devices.

The Planner

A goal is input to the planner as a description of a desired event. The concept associated with such an event can have execution knowledge attached to it; the planner uses that knowledge. If no execution knowledge exists, and background knowledge fails to solve the problem, then the planner signals a failure.

The Temporal Database of Events

The temporal database has some of the features of a time map management system.[2] It maintains a temporal partial order of all events, and can determine the temporal relationship between any two events, including the temporal distance between them.

Conceptual Knowledge

Conceptual knowledge embodies the system's understanding of the universe. It is divided two ways: a concept is either a descriptor or a rule, and either an hypothesis or a theory.

Descriptor concepts represent attributes, such as temperature or acceleration. Primitive descriptor events are sensor inputs or actuator commands. Values of derived descriptor events are computed from the values of other events; the knowledge of how to do these computations is part of the descriptor concept. Knowledge attached to each descriptor includes the range of observed values.

Rule concepts make predictions in the form, "the presence of a certain type of event implies the presence of another type of event with a specified temporal relationship to the first event". A rule event is a pair of descriptor events which satisfies the rule. Each rule has knowledge of its counter-examples, i.e. events which satisfy the rule's premise but not its conclusion.

A rule hypothesis is a rule which has not been confirmed often enough to be considered reliable, but which is not obviously false. For example, it might have only one observed example and no counter-examples. The verifier knowledge source decides when a rule hypothesis is reliable enough to be a theory, using criteria in the verifier's background knowledge. An example criterion is that the concept must have at least three examples, and the number of counter-examples must be less than 10% of the number of examples. The refuter knowledge source decides when a rule hypothesis is so unreliable that it should be forgotten, e.g. when there are at least three counter-examples, and the number of counter-examples is at least 30% of the number of examples.

A descriptor hypothesis is a descriptor which is not yet known to be useful. A descriptor can be useful in three ways: it can be an observed constant, it can be a term in a rule theory, or it can be a sub-concept of one of these. Our use of observed constants for learning is akin to BACON.[4] The verifier uses background knowledge to decide when a descriptor is a reliable and useful constant. Sub-concepts of constants or rules automatically become theories when the concepts they support become theories.

The Learner

The conceptual structure of the learner shown in Figure 2 comprises a topic selector, a hypothesis tester, a surprise monitor, and an induction system.

The topic selector decides what theory to investigate. Evaluation criteria include the number of applicable hypotheses and the relationship to unmet goals. The most promising theory becomes the topic. The planner tries to produce an example event, called the topic event.

The hypothesis tester determines whether the topic event acts to support or refute any hypotheses. If supportive, the event and the hypothesis are passed to the verifier, which may promote the hypothesis to a theory; otherwise, they are passed to the refuter.

The surprise monitor notices and tries to explain events that defy expectations. For example, a new event may dramatically extend the observed range of a descriptor. The surprise monitor would search for events which might have caused this change, e.g. the first event of an actuator descriptor. If a plausible cause is found, the surprise monitor forms a rule hypothesis (and necessary supporting descriptors) to predict such cause-effect pairs in the future.

The major elements of the induction system are the partner selector, the example bounder, and the induction element. The partner selector heuristically chooses a descriptor to be paired with the topic concept. The example bounder pairs individual events of the topic with events of the partner. These event pairs serve as examples for the induction element, which uses constructive generalization rules to find hypotheses relating the topic concept to the partner concept.

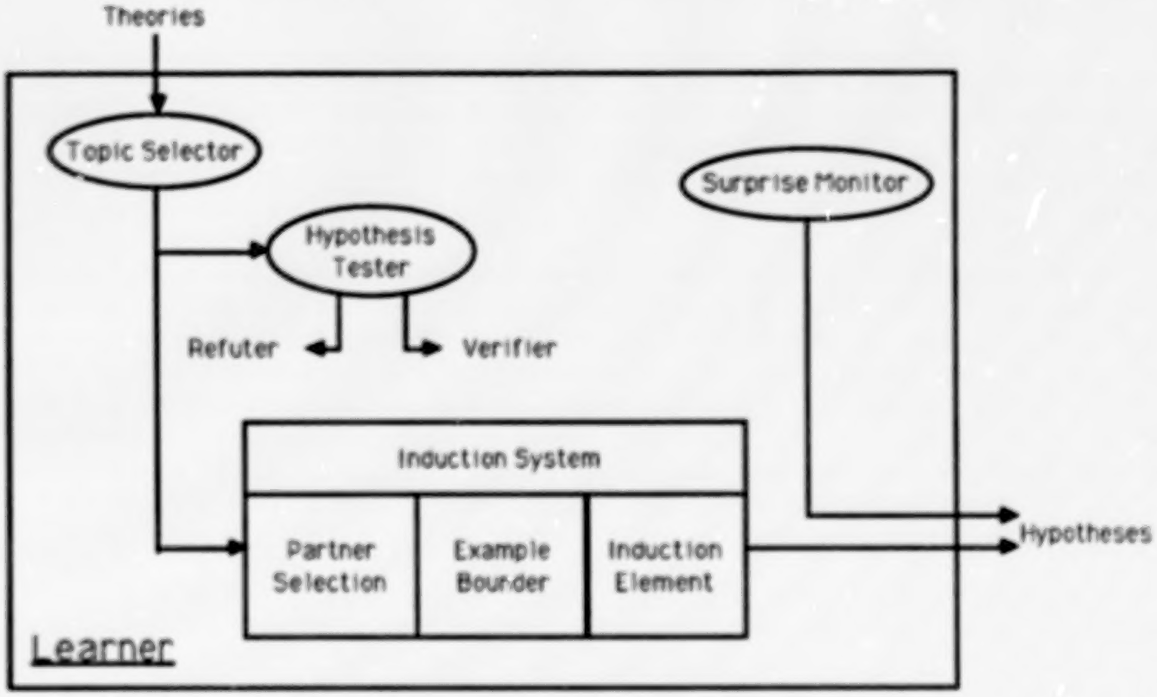


Figure 2. Learner Components

IMPLEMENTATION AND TEST

We have implemented a simple version of the proposed architecture in the general purpose autonomous learner (GPAL) program. GPAL has been applied to learning about the world of a mobile robot.

GPAL is implemented in Common LISP and remotely controls a Heath/Zenith HERO 2000 mobile robot. Primitive descriptors known to GPAL at start-up are:

1) Send out a SONAR pulse.
2) MOVE the robot forward or backward some distance.
3) Sense the DISTANCE to a wall in front of the robot.

Unknown to GPAL, SONAR events cause DISTANCE events; DISTANCE events cannot be instantiated alone. The robot faces a wall, so the value returned by DISTANCE changes in proportion to the value of MOVE events.

An annotated trace of a GPAL run is listed below. Comments are in italics. Events are shown by name, e.g. EVENT-2071, and by content, e.g. (DISTANCE 90). For clarity and brevity, no individual events are shown after cycle 3.

```
> (gpal)
EVENT-2064 (SONAR)
EVENT-2071 (DISTANCE 90)
"Topic-event: (SONAR)"
"Making rule SONAR-DISTANCE-UNEXPECTED"
"Making rule DISTANCE-SONAR-UNEXPECTED"
End cycle 0
```

In cycle 0, GPAL studies SONAR. A SONAR event is created, and a DISTANCE event occurs as an unexpected side-effect. GPAL makes two rule hypotheses from this: 1) That a DISTANCE event will always immediately follow a SONAR event, and 2) that a SONAR event will always immediately precede a DISTANCE event.

```
EVENT-2121 (SONAR)
EVENT-2124 (DISTANCE 91)
"Topic-event: (SONAR) (SONAR)"
"Unexplained events since topic-event: (EVENT-2124)"
End cycle 1

EVENT-2129 (SONAR)
EVENT-2130 (DISTANCE 90.5)
"Topic-event: (SONAR)"
"Verifying SONAR-DISTANCE-UNEXPECTED"
"Verifying DISTANCE-SONAR-UNEXPECTED"
"Inductive partner: DISTANCE"
End cycle 2
```

In cycles 1 and 2, GPAL tests the hypotheses about SONAR and DISTANCE. In cycle 1, the DISTANCE event is unexplained because its value of 91 is outside the previously known range of [90]. There is no obvious cause for the extension, and the new value is incorporated into the known range of DISTANCE. In cycle 2, enough examples have been gathered to verify the rules relating SONAR and DISTANCE. The hypotheses become theories. GPAL naively tries to induce a relationship between DISTANCE values and SONAR; the example bounder forms examples from event pairs which are examples of the learned rules.

```
EVENT-2148 (MOVE -20)
EVENT-2149 (SONAR)
EVENT-2150 (DISTANCE 109.5)
"Topic-event: (MOVE -20)"
"Inductive partner: DISTANCE"
"Unexplained events since topic-event: (EVENT-2150)"
"Making descriptor DELTA-DISTANCE"
"Making descriptor ABS-DELTA-DISTANCE"
"Making sub-range SR1/2-ABS-DELTA-DISTANCE"
"Making sub-range SR2/2-ABS-DELTA-DISTANCE"
"Making rule MOVE-SR2/2-ABS-DELTA-DISTANCE-EXCEPTION"
End cycle 3
```

In cycle 3, GPAL picks MOVE as the topic. It tries to relate the value of MOVE to the value of DISTANCE, so a new DISTANCE event is created (using learned knowledge of how to do this). No temporal rule links MOVE and DISTANCE, so the example bounder pairs events by temporal proximity.

The new DISTANCE event has a value far outside the previously observed range. A new operator, MOVE, has been used, so the surprise monitor attributes the range extension to MOVE. New descriptors are created: change in DISTANCE, size of change in DISTANCE, small (SR1/2: x <= 2.0) and large (SR2/2: x >= 2.0) size of change in DISTANCE. These are used in the rule hypothesis that "each large-change-in-DISTANCE event temporally contains a MOVE event".

```
"Topic-event: (MOVE 4)"
"Inductive partner: DISTANCE"
End cycle 4

"Topic-event: (DISTANCE 105)"
End cycle 5

"Topic-event: (DISTANCE 99.5)"
End cycle 6

"Topic-event: (MOVE -13)"
"Verifying MOVE-SR2/2-ABS-DELTA-DISTANCE-EXCEPTION"
"Verifying SR2/2-ABS-DELTA-DISTANCE"
"Verifying ABS-DELTA-DISTANCE"
"Verifying DELTA-DISTANCE"
"Inductive partner: ABS-DELTA-DISTANCE"
End cycle 7
```

MOVE and DISTANCE events are generated in cycles 4-7. In cycle 7, the verifier is convinced that the rule linking MOVE to large changes in DISTANCE is true. The rule becomes a theory, which promotes its supporting descriptors to theories.

Cycles 8 through 23 are fruitless attempts to find other hypotheses. We resume the example at cycle 24.

```
"Topic-event: (MOVE -19)"
"Inductive partner: DELTA-DISTANCE"
"Making descriptor DELTA-DISTANCE-/--1.028"
"Making descriptor DELTA-DISTANCE-/--1.028-MINUS-MOVE"
End cycle 24
```

Here MOVE is inductively matched with DELTA-DISTANCE. The example bounder pairs events of the two descriptors, using the known temporal link between MOVE and SR2/2-ABS-DELTA-DISTANCE. An inductive knowledge source finds a noisy linear relationship between the two. This is represented by new descriptor hypotheses. The first is DELTA-DISTANCE divided by the slope of a DELTA-DISTANCE vs. MOVE graph; the second is this value minus MOVE, which is equivalent to the y-intercept divided by the slope. The second descriptor will be nearly constant if the linearity holds. There are previous examples, but the verifier requires at least one more example after an hypothesis has been proposed.

```
"Topic-event: (DELTA-DISTANCE -1)"
"Inductive partner: DISTANCE"
End cycle 25

"Topic-event: (SONAR)"
"Inductive partner: SR2/2-ABS-DELTA-DISTANCE"
End cycle 26

"Topic-event: (ABS-DELTA-DISTANCE 0)"
End cycle 27

"Topic-event: (SR2/2-ABS-DELTA-DISTANCE 19.5)"
"Inductive partner: DISTANCE"
End cycle 28

"Topic-event: (MOVE -6)"
"Verifying DELTA-DISTANCE-/--1.028-MINUS-MOVE"
"Verifying DELTA-DISTANCE-/--1.028"
"Inductive partner: DISTANCE"
End cycle 29
```

In cycle 25, GPAL tries to create a DELTA-DISTANCE-/--1.028-MINUS-MOVE event, but fails; the new descriptor is neither verified nor refuted. In cycle 26, a new example is created, but GPAL is only looking at super-concepts of the topic, SONAR. A similar missed opportunity comes in cycle 28. In cycle 29, GPAL returns its attention to MOVE, makes a new DELTA-DISTANCE-/--1.028-MINUS-MOVE event, and verifies that the descriptor is relatively constant. Both descriptor hypotheses are promoted to theories. Aside from slight numerical adjustments, GPAL now understands everything in its environment.

DISCUSSION

We have described the example bounding problem, and claimed that solving this problem is a key to building autonomous machines that learn and are useful. We have proposed a machine architecture which incorporates an example bounder. The architecture has been implemented in the GPAL program.

Tests of the GPAL program with a mobile robot show that it is able to learn the rules governing a simple real environment. The example bounder plays a vital role, forming the examples from which GPAL learns the linear relationship between MOVE and DELTA-DISTANCE. GPAL's gradual learning of the relationship between MOVE and DISTANCE shows a conceptual progression from qualitative physics to quantitative understanding. Early on, GPAL learns that MOVE causes large changes in DISTANCE. Later, it quantifies that relationship. Both sets of knowledge are retained and are available to the planner.

It remains to be shown that the proposed architecture is adequate for general robot learning in complex environments. We will incorporate real-time visual information and two-dimensional motion in the future. Such tests require more heuristics for learning so that GPAL can recognize more subtle regularities in its environment. Greater noise adaptation is particularly important; the heuristics used now sometimes cannot handle the noise of the current environment.

We have barely addressed forgetting, an issue which will gain importance as learning machines with finite memories begin to learn over long times and many domains. Our architecture addresses only the forgetting of refuted hypotheses. Forgetting of events and of easily re-learned concepts will need to be addressed eventually.

GLOSSARY

We use the following definitions:

Attribute: a measurable or observable feature, e.g. weight or color.
Complex event: an event comprising two or more simple events, used as input to an inductive learner.
Concept: a rule or a descriptor.
Derived concept: a concept which is defined in terms of one or more other concepts.
Instantiation: creation of the data object which represents a simple event of a specified concept.
Primitive: a descriptor defined by the machine in which the system runs. The instantiation side effects of primitives are the means by which the device interacts with the world.
Simple event: a single observation of one attribute, or an example illustrating a law.
Sub-concept: one of the concepts from which a derived concept was derived. A primitive has no sub-concepts.
Super-concept: a derived concept is a super-concept of concepts from which it is derived.

REFERENCES

1. Anderson, J.R., "Causal Analysis and Inductive Learning", Proc. of 4th International Workshop on Machine Learning, 1987, 288-299.

2. Dean, T.L., and D.V. McDermott, "Temporal Database Management", Artificial Intelligence 32, 1987, 1-56.

3. Dieterrich, T.G., and R.S. Michalski, "Discovering Patterns in Sequences of Events", Artificial Intelligence 25, 1985, 288-299.

4. Langley, P, G.L. Bradshaw, and H.A. Simon, "Rediscovering Chemistry with the BACON System", in Machine Learning, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, eds., Tioga, Palo Alto, 1982.

5. Tillotson, B., C. Lin, and J. Bezdek, "Creating Input Sets for Inductive Learning from Simple Events", Proc. of SPIE Applications of Artificial Intelligence VI, 1988, 273-277.

N89 - 15589

TOWARD A COMPUTATIONAL THEORY FOR MOTION UNDERSTANDING: THE EXPERT ANIMATORS MODEL

Ahmed S. MOHAMED
William W. ARMSTRONG

Department of Computing Science
The University of Alberta
Edmonton, Alberta, Canada T6G 2H1

ABSTRACT

Artificial intelligence researchers claim to "understand" some aspect of human intelligence when their model is able to "emulate" it. In the context of computer graphics, the ability to go from motion representation to "convincing" animation should accordingly be treated not simply as a trick for computer graphics programmers but as important epistemological and methodological goal. In this paper we investigate a unifying model for animating a group of articulated bodies such as humans and robots in a three-dimensional environment. The proposed model is considered in the framework of knowledge representation and processing , with special reference to "motion" knowledge. The model is meant to help setting the basis for a computational theory for motion understanding applied to articulated bodies.

[1] INTRODUCTION

Articulated body movements cannot be understood by using the current techniques of computer animation even if they incorporate dynamics to model every detail. There is a significant difference between "understanding" the motion and "synthesizing" it. This is the difference between the point of view of "cybernetics" and the point of view of "computer graphics". The latter starts from a set of trajectories or forces and torques acting on a body, and tries to account for the resulting movement: "how is the arm going to move if the hand has to follow such a trajectory?", "how is the body going to react if the legs are exerting such a force on the ground?".

The point of view of cybernetics is just the opposite: it starts from the definition of the goal (a desired behaviour) and then tries to force the system to follow it, possibly using a large range of current motion control techniques, either of the feedback or feedforward type. The emphasis, in this case, is on the knowledge that is required to produce fluent natural motion performance: sensory knowledge, motor knowledge, knowledge processing, abstract representation of knowledge structures. These are all incorporated in our term "motion" knowledge.

In AI terminology, knowledge representation means defining concepts and rules able to capture the essential complexity of a given problem domain that escapes a direct approach [20]. Thus, a knowledge representation approach emphasizes the "goals" and the "functions" of a system rather than its specific mechanisms, and in so doing it generalizes and abstracts our understanding of the system, since it is likely that different mechanisms can be found that can implement the same function and goal.

Although knowledge representation and processing are "hot" topics in artificial intelligence research, very few attempts have been made to help understand motion, which means to build up a model of "motion" knowledge. In this paper, we concentrate on the knowledge aspects of motion and propose a unifying model for animating a group of articulated bodies that we hope will be useful for investigators who are dealing with articulated body animation in the large. This includes neuroscientists, biomechanists, dance designers, motor rehabilitators, computer animators, and robotics researchers. It is believed that all these researchers basically require the same type of motion knowledge whether they are writing a computer script for dance ballet, a program for figure animation, a control program for simulated robotic manipulator arm, or describing with a symbolic motor script the movements of a child or of a motor program for rehabilitation [16].

To make our point about the need for motion knowledge clear, let us look at "music" which is a complex phenomenon similar to motion from many viewpoints. The situation is quite different for music. Indeed, Camurri [5] showed how music notation, as an example of a symbolic representation of a complex phenomenon, was successfully able to discriminate which aspects of complexity to represent explicitly and which to represent implicitly. This successful notation was able to capture the essential structure of music (what in AI terms could be called "music" knowledge). It is easy to notice the abstractness and functionality of the notation, for example the instruments, are not shown in the notation, nor the way in which a performer plays a specific instrument, and the individual style of performance. This adequacy of the music notation symbolism to express the essential structure of music is clearly demonstrated by its ability to survive the advent of computer era.

Music notation can be easily expressed in computer terms and can be used directly to drive computer music synthesizers [3] [17]. For dance and movements, on the contrary, the picture is quite different. Dance and movement notation methods have proliferated, without finding the same success as music notation, and computer techniques directly applied to them do not seem so far to pass the basic test: the ability to generate from the notation a fluent, natural ("convincing") synthetic motion performance [4] [22] [24].

In this paper our principle intention is not to look for yet another movement notation or language for motion, but to tackle motion from a broader prospective. We basically are trying to produce "convincing" animations for a group of articulated bodies such as humans and robots without pressing the animator to become overly involved in the mechanisms of producing the motion. We intend to shift this burden from the animator to the individual articulated bodies through developing an expert animator agent for each member of the collection articulated bodies. These agents have a computational understanding of motion and its semantics in a way that each individual articulated body would handle its motion autonomously. They also have the capabilities to communicate with each other and with the animator. The model allows each agent to have its own behaviour depending on its specific role in the group, duties, areas of responsibilities, etc.

In previous work [14] [15], we have developed a simulation for a multi-legged articulated robot that could be useful in constructing and maintaining structures in space stations such as solar arrays, large multibeam antennas, and space factories. The robot used its legs both for locomotion and for object manipulation. From our review of space literature, we sometimes noted a requirement for more than one robot, since many tasks can only be perfomed through cooperation of multiple robots. In a sense, we felt that the multi-robot simulation would be a natural extension to our single robot simulation. But, the multiple robot extension brought up two research issues that do not appear in the single robot problem:

(1) The method we used for planning the motion for the single robot was based on the assumption of a static environment, and so it can not be used in the multiple robots case because each of the robots is in a dynamic environment consisting of other moving robots;

(2) The technique we used to animate a single robot's motion was based on calculations of the dynamic equations of motion that were executed on a distributed processor in order to produce the motion in real time. Producing the motion dynamically for the multiple robot case would need much greater processing capability that is currently unavailable.

In order to overcome these two problems and to produce "convincing" animations (as opposed to simulations) for the group of robots without pressing the user to become overly involved in the mechanisms of producing the motion , we have developed an expert animator agent for each robot. Each agent integrates knowledge engineering approaches, namely, object-oriented programming and rule-oriented programming [19] [25] with computer animation approaches. The object-oriented approach plays a key role in the modeling of the robots' inter-relationships, whereas the intelligent functions of each expert animator agent are transparently programmed in rule-oriented programming style. In producing animations for the various robots, each robot is considered to be an object in the environment. It handles its motion and interactions with other robots as well as with the animator autonomously. To program each expert animator agent intelligently in order to make it adapt to its environment, we adapted the method of production systems.

In order to integrate a rule-oriented approach with an object-oriented approach, the concept of ruleset of "LOOPS," a programming system developed at Xerox PARC, was very instructive [25]. A ruleset is a sort of object which consists of ordered rules with specified control structures for selecting and executing the rules. Each agent keeps several rulesets for performing locomotion, avoiding obstacles, deciding task priorities, etc.

Convincing animations here means that each expert animator agent should be able to maintain the following motion requirements: (1) produce sustained stable locomotion, i.e, maintain its robot orientation, have control over its velocity, and avoid obstacles, (2) choose the most appropriate locomotive skill at any point during the robot's navigation (e.g, walk, trot, climb, etc.), (3) deal intelligently with the environment (Winkless [29] has defined intelligent locomotion as ".. the ability to do appropriate movements under unpredictable conditions"), (4) maintain the robot's static and dynamic stability, (5) ability to combine different locomotion skills (e.g, turning while running), (6) achieve smooth transitions between different locomotive skills, (7) prefer the paths the robot has traveled on before, (8) reduce total energy consumption in executing the robot's missions, (9) perform the robot's task-specific operations elegantly, (10) cooperate with other robots in the environment-either avoid colliding with any of them or cooperating with them in any multi-robot task.

In order for each expert animator agent to satisfy all these requirements, each treats motion in a cognitive framework analogous to co-operation among several motion processes. During motion production, these processes appear or disappear, modify or repeat themselves. Moreover, each expert animator agent has a "motion" knowledge base that provides several levels of sophistication (multiplicity of expressive systems). These levels along with the motion processes are embedded in a formal framework that permits animating the motion at different levels of detail. Each expert animator agent controls its robot's motion at a descriptive level appropriate to the context of its motion. Thus as long as an expert agent determines that its context needs a simple style of movement to produce "convincing" animations, simple techniques will be selected by the agent. As soon as any agent concludes that its context needs more natural, coordinated, task-oriented, and expressive motions, the agent becomes more sophisticated and increases its descriptive level of motion control.

The power of the expert animator agents lies in their generality and "cognitive penetrability" [18] to variations of the basic motion patterns such as uniform trotting, walking over obstacles, overcoming obstacles, etc. In other words the expert agents are able to manage the environmental disturbances without any ad hoc modification to their methodology in motion production. Their formalization power is generic enough to adapt to these disturbances. Moreover, the agents also take care of the robots' interactions with each other. Each expert animator agent optimistically executes each robot's plan without taking into account the existence of other moving robots. Then when two or more robots detect the danger of collision, they negotiate to refine their global path plans in order to avoid collision.

Relying on their "motion" knowledge bases the expert animator agents employ several levels of reasoning in both their negotiations and their answers to the animator's explanation enquiries: (1) Geometric reasoning : both static (e.g, "Am I now on top of obstacle o_i?") and dynamic (e.g, "Can I use the robot's left front leg to reach for the tool t_1 and grasp it?"), (2)Common-sense reasoning (e.g, "Should I switch now to running?", "Should I allow one of the conflicting robots to go first or ask to allow me to do so?").

Actually the multi-robot problem has been investigated in AI under "Distributed Artificial Intelligence (DAI)" where multiple intelligent agents are presented with a complex problem solving situation. Various aspects of DAI research could be found in [6] [7] [8] [26].

The problem has also been investigated in theoritical studies of motion planning for multiple moving objects under "the Piano Movers problem": the problem of planning the motions of several objects among polygonal obstacles. Various aspects and special cases of the research problem could be found in [27]. In this paper we tackle the problem from the graphics animation angle, setting our goal to produce "convincing" animations that satisfy the aforementioned requirements.

In Section 2 the expert animators model is described in the context of our multi-robot space station environment. Section 3 presents a simple experimental system to demonstrate our model. Our conclusions appear in Section 4.

[2] THE EXPERT ANIMATORS MODEL

Figure 1 shows our multi-robot environment. Each robot has its own behaviour that depends on its specific tasks in the space station. For example we have satellite expert robots, mechanical expert robots, electrical expert robots, etc. Each has its own predefined set of control routines for its specific tasks. All the robots are "physically" alike (see figure 2), they are articulated with four legs each and 18 degrees of freedom. We obtained a crude estimate of the number of degrees of freedom (DOFs) which are needed for free locomotion of each robot by observing that during locomotion it must ideally be possible to control the six DOFs of the body (three translational and three rotational) when it is supported by each of the two alternating sets of legs(at least two legs should be in contact with the station ground all the time to achieve static stability). So one might expect about twelve DOFs to be a minimum for the four-legged robot. If twelve DOFs are taken as a rough estimate, they can be distributed among the robot's four legs as three DOFs each. The robots of figure 1 have three DOFs for each of their four legs: two at the hip (one for elevation and another for lateral movements) and one DOF at the knee.

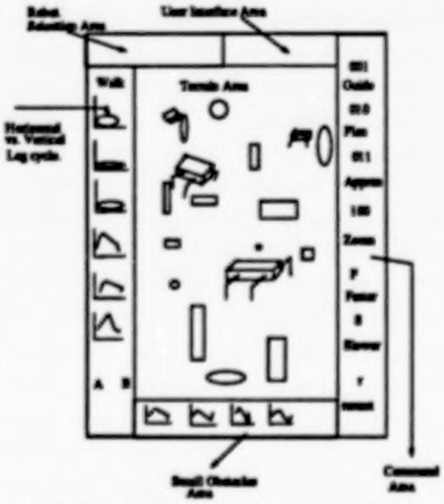


Figure 1: The Multi-Robot Expert Animator System

The robots "live" in a space station that contains obstacles such as blocks, holes, inclines, declines, and rough terrain. Some of the obstacles in the environment are small so the robots do not "see" them until they are navigating close to them. In such cases, the robots have to modify their motions on the fly to avoid or overcome such obstacles upon "perceiving" them. It is important to realize that the robots have no control over what they will encounter in the environment before actual motion execution. Winkless [29] defined intelligent motion as ".. the ability to do appropriate motion under unpredictable conditions". Thus, a preprogrammed motion, highly accurate, productive, precisely measured and well understood in each robot cannot be considered an intelligent one, since a robot has no ability to cope with unpredictable situations and to choose between alternatives. The robots have onboard cameras fixed on top of them. These cameras regularly feed to navigation systems the local obstacles that each robot faces. The traversals of the robots are based on a local navigation strategy that use the on-board camera information [13] [14]. Each robot is capable of four types of tasks: (1) vision-related tasks: scan the surrounding environment- turn the camera 180^0- tilte the camera θ^0 left- align the nearest object to the camera, etc. (2) locomotion tasks: walk, trot, run, turn-left, turn-in-place, stop, etc. (3) task specific operations: grasp, nock, screw, etc. (4) negotiation-related tasks: either to avoid colliding with other robots during navigation, or to co-operate with others in performing multi-robot tasks. The situation we are dealing with here is characterized by the following features: the desired motion trajectories are frequently unknown; the environment is described vaguely (because of the existence of unknown small obstacles); the robots are highly nonlinear, coupled, and redundant. Under these conditions, the robots' expert animator agents are required to produce purposeful motions ("convincing animations") in real time for all robots.

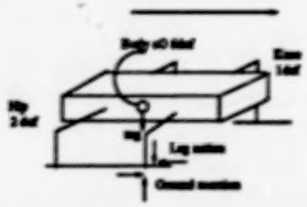


Figure 2: The Four-legged Articulated Robot

Figure 3 shows the internal structure of an expert animator agent. The expert animator agent consists of the following modules: (1)The task planning and execution monitoring module, (2) The agent model, and (3) the agent reasoning module. In the following we describe each in some details.

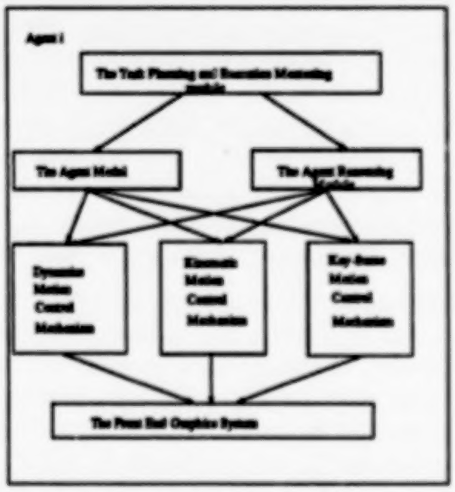

Agent i
The Task Planning and Execution Monitoring Module
The Agent Model
The Agent Reasoning Module
Dynamic Motion Control Mechanism
Kinematic Motion Control Mechanism
Key-frame Motion Control Mechanism
The Front End Graphics System

Figure 3: An Expert Animator Agent Internal Structure

[2.1] TASK PLANNING AND EXECUTION MONITORING MODULE

This module generates and executes plans for carrying out tasks that a robot could be performing at any specific time either individually or simultaneously along with other robots. The module consists of a planner and an executor. The planner has knowledge of the large obstacles and objects in the environment; their relative locations and object capabilities. The planner takes an input a task submitted by the animator, or a request from the executor to replan the current task. The planner produces as output a sequence of plan steps to be executed by the different executor routines. For example the task: "Get tool t_i and convey it to robot R_j" could produce the following plan: (1) to the Navigator: "Go to tool box b_k" (2) to the Relative Referencing: "Line-up on the right side along tool box b_k" (3) to the Local Obstacle Avoidance: "Follow the side of the tool box" (4) to the Pattern Matcher: "Identify tool t_i in the tool box- convey its position and orientation (x_1, y_1, θ_1)" (5) to the Trajectory Generator: "Compute a path from a suitable leg/arm to (x_1, y_1, θ_1)" (6) to the Gripper: "Grasp tool t_i at (x_1, y_1, θ_1) using arm AR_k" (7) to the Leg/Arm: "Place tool t_i on top of the robot's body" (8) to the Navigator: "Go to robot R_j" (9) to the Local Obstacle Avoidance: "Follow the side of Robot R_j" (10) to the Trajectory Generator: "Compute a path from a suitable Leg/Arm to R_j's body top" (11) to the Gripper: "Grasp tool t_i from top of the robot's body" (12) to the Leg/Arm: "Place tool t_i on top of robot R_j's body"

Each step in the previous plan is called a plan step. The planning technique that is used by the planers does not concern us here [13]. Upon receiving message of a plan step completion from one of its routines, the executor checks out the success of the step. If the step executes normally the executor proceeds to the next plan step. Otherwise the executor reacts to any abnormal conditions by sending orders to carry out corrective actions and wait for the next solicitation.

[2.2] AGENT MODEL

The agent model provides several levels of sophistication (a multiplicity of expressive systems) to describe the associated robot's state and its surrounding environment. These levels permit animating the robot's motion at different levels of detail. The agent model contains three levels of descriptions: (1) Conceptual level; (2) Topological level and (3) Dynamics level. Different motion control mechanisms work on these levels. They are, respectively, (1) key-frame motion control mechanism; (2) Kinematics motion control mechanism; (3) Dynamics motion control mechanism. Each expert animator agent views the different levels of descriptions and their manipulation mechanisms as Frame structured data [19] called objects. In object-oriented programming, information and its manipulation mechanisms are put together and represented in the form of objects. Figure 3 shows the different objects that the expert animator deals with. The objects' mechanisms are contained in the agent's reasoning module whereas the robot's descriptive levels are contained in the agent model.

The conceptual level describes the associated robot's capabilities and responsibilities. This includes behaviour characteristics, duties, areas of responsibility, role in a group, etc. Also, general properties of the robot may be expressed here, such as the hands (legs) being used for most grips, the relationship between the size of the object gripped and the capacity of the gripper, etc. The task-specific functions of each robot are represented in a data structure similar to what Turvey [28] called action concepts or what Schank called primitive actions in his Conceptual Dependencies [23] (semantic structure of actions or action verbs in the fields of psychology and linguistics). Any plan step (the output of the Task Planning and Execution Monitoring module) is defined in terms of interrelated component actions, e.g, "reach" for tool t_i, "lift" and "transport" tool t_i above the tool box b_k such that it can be "lowered" into the top opening of the box.

For example Figure 4 represents the plan step for robot R_1 "Use the Screwdriver S_i to screw a screw in the wall". The diagram represents the action of the plan step at its highest node by Agent (x=R_1) screw Object (y=screw) with Implement (z=Screwdriver). The node "screw" includes three semantic subpredicate nodes, each of which stands for a distinct relational concept. An arrow originating from a given node terminates on an entity that is linked through the relation expressed at the arrow's node of origin. Thus, the two predicates "Move" and "Motion" are the arguments for the predicate "Cause", and the labels "event" and "result" indicate the role that "Move" and "Motion" play, respectively, in relation to "Cause". What the Figure represents is that Agent R_1's hand (h_2) movement is an event that causes Implement z (Screwdriver) to move with respect to object y (screw) in a certain spatiotemporal manner. Additionally, the hand and screwdriver are related as implement and object, respectively, through the "Grasp" node; and the screw is located (L) with respect to the tool box b_k through the "Contained In" node.

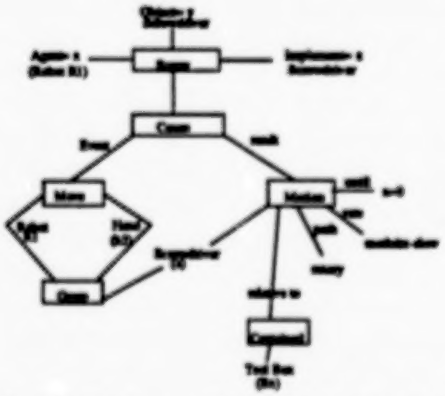


Figure 4: Semantic Representation of the plan step "Use the Screwdriver Si to screw a screw in the wall".

This representation is an abstract specification of the plan step in terms of desired relationships between the robot and objects in the environment. Such relationships are expressed functionally, spatially, and temporally with respect to the action situation. For example, the rotary motion of the screwdriver is specified relative to the wall; however, the precise trajectory of the screwdriver, the shapes of the screwdriver and tool box, and the exact environmental location and orientation of them remain indefinite. What is important is that the representation at this level involves only those spatial characteristics that allow identification of objects and actions as well as action-relevant properties of objects. In a similar way, the temporal parameters of a movement are not defined precisely at this level, but are stated in rather qualitative terms, e.g., the "medium-slow" motion of the screwdriver.

The agent reasoning module (see below) makes use of such conceptual level descriptions of plan steps in two ways: (1)to build key-frames for the associated robot and its surrounding objects to be used as the input for the key-frame motion control mechanism; (2)to provide semantic reasoning through token propagation in the semantic diagram. The purpose of the reasoning here is either to answer the animator's questions about the associated robot's behaviour, or to provide means of high-level negotiations with other robots.

The topological level associates a coordinate frame with each robot's limb and objects in the robot's surrounding environment. Furthermore frames are grouped together to refer to some structures (a robot, an object, etc.). The coordinate frames make us view the robot's actions as streams of variations of some of the mutual relations between the coordinate systems which are due to the stream of motion commands. At this level the expert animator agents view everything as a "forest" of coordinate frames that change over time. By visiting the forest it is possible to express the geometric relations between any two coordinates in the system. This is what in the expert animator agent's reasoning module is called topological reasoning (see later). Figure 5 shows how a coordinate frame is associated with each robot limb and each object in the robot's surrounding environment. In order to perform the plan step explained above ("Use the screwdriver to screw a screw in the wall") the Agent R_1's hand (h_2) frame should overlap with the coordinate frame of the screwdriver and then rotate it with respect to the wall, the screw's frame should overlap with the wall's frame, etc.

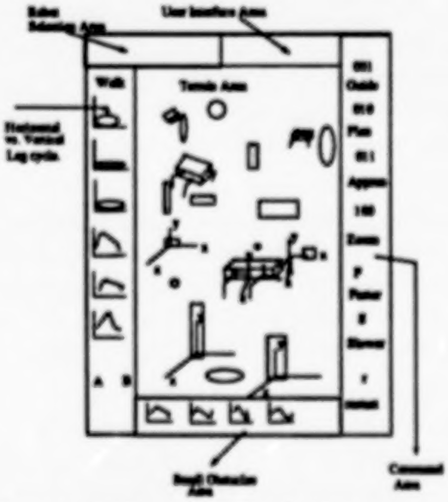


Figure 5:The Geometric level of the agents.

Again the agent reasoning module will make use of such a topological-level description of plan steps in two ways: (1) to provide a framework for the kinematic motion control mechanism (see next section), and (2) to provide topological reasoning either to answer the animator's questions or to interact with other robots.

The dynamics level describes the dynamic properties that are required by the dynamics motion control mechanism in order to perform any plan step dynamically. This includes the masses, locations of centers of masses, moments of inertias, joint spring and damper values, etc.

[2.3] AGENT REASONING MODULE

The agent reasoning module has the responsibility of deciding what are the best mechanisms for producing "convincing" animations for the associated robot in the current motion context. The agent reasoning module takes the planning steps from the Task Planning and Execution Monitoring module and animates the actions at the appropriate sophistication level. The reasoning module evaluates the current context that the associated robot is in (e.g, where the animation camera is, whether the robot is involved in a multi-robot task, whether the robot is out of sight in the current scene, etc.). As long as the agent's reasoning module evaluates that the associated robot's context requires a simple style of movement, simple techniques (e.g. key-frame motion control mechanism) will be selected to drive the associated robot. As soon as the reasoning module determines that the context needs more natural, coordinated, task-oriented, and expressive motions, the reasoning module request the robot to be more sophisticated and increase its level of motion control.

We adapted rule-based programming for describing the various control and decision mechanisms of the agent reasoning module. In order to integrate this rule-oriented approach with the object-oriented organization of the expert animators, rules are organized in rulesets [25]. A ruleset consists of ordered rules with specified control structures for selecting and executing the rules. One such ruleset is the sophistication-level ruleset. Some of its rules are:

> IF THE ROBOT IS OFF-SIGHT IN THE NEXT FRAME THEN SET THE CONTROL SWITCHES TO NULL.
>
> IF THE ROBOT IS CLOSE TO THE CAMERA THEN SET THE CONTROL SWITCHES TO DYNAMICS MOTION CONTROL MECHANISM.
>
> IF THE ROBOT IS ENGAGED IN A GROUP TASK THEN SET THE CONTROL SWITCHES TO MIXTURE OF DYNAMICS-KEYFRAME CONTROL MECHANISMS.
>
> IF THERE ARE ANY INTERESTING ACTIONS THEN MOVE THE CAMERA CLOSER TO THE ACTIONS.

The control switches mentioned in the rules are data structure in the reasoning module (see later).

The agent reasoning module has also to provide an explanation-based interface to the animator. Each expert agent should be able to answer the animator's questions about any of the decisions it has taken. For this purpose, the agent reasoning module uses two reasoning mechanisms: (a) Semantic reasoning through token propagation at the conceptual level of the agent model, (b) Geometric reasoning at the geometric level of the agent model.

The Semantic reasoning mechanism answers relationship questions among objects by spreading activation out from each of the objects nodes and seeing where the activations meet [20]. Using this mechanism, it is possible for the agent reasoning module to use a diagram such as the one in Figure 4 to answer questions such as "what is the relationship between the robot R_1 and the screwdriver s_i?". By spreading activation from both the robot R_1 and the screwdriver, the activation meets at the two nodes "Grasp" and "Screw".

The Geometric reasoning mechanism answers geometric types of questions. For example "Are you holding the screwdriver now?", or "Can you reach for place (x_1, y_1) on the wall?". The answer for the first question is "yes"if one of the robot's gripper frame is on top of the screwdriver frame. For the second question the geometric reasoning mechanism will calculate the distance between the frames and access the robot's arm reach capabilities from the agent model.

Finally, the agent reasoning module has the responsibility for navigating its associated robot safetly in the environment. The navigation problem is decomposed into two subproblems: a global path planning problem and a local path planning problem. Each robot's agent reasoning module plans independently its own path for each plan step from its initial location to the final location. No positional constraints introduced by any other robot nor any small obstacles are considered. This is called the global path planning level. At this level the agent reasoning modules use the conventional motion planning solution for the single articulated robot case [15]. However each agent reasoning module will revise its global plan in two situations:

(a) Whenever there is conflict between path plans,i.e, a collision between two or more robots may occur. Robots R_i and R_j are said to be on a collision course between the period t and $t+\delta t$ if

$$dist(R_i,R_j)\leq(v_i+v_j)\delta t+d$$

where: v_i is the velocity of the robot R_i and v_j is the velocity of robot R_j. d is the safety allowance distance between any two robots. In such cases, domain-specific knowledge describing the robots' current situations is usually used to resolve this conflict. This may include such information as the urgency of one to reach a goal location, the degrees of freedom available for modifying the planned path, the interdependency between the sub-tasks to be performed by the involved robots, etc. All these data are deduced by the agent reasoning modules from their corresponding agent models. The robots bargain with each other through a process of exchanging knowledge about their situations and suggesting plan revisions. Each agent reasoning module stores some rules for evaluating the descriptions of their relative priorities (this is called the priority ruleset). Example of one such rule is:

> IF MY MISSION IS OF PRIORITY=HIGH AND MY TASK STATUS=NEAR-COMPLETION AND NONE OF THE INVOLVED ROBOTS HAS PRIORITY=RUSH THEN ASK FOR THE RIGHT OF THE ROAD.

A robot can start to replan its path and resume its motion if no other robots of higher priority are at a distance shorter than the safe interrobot distance (d). This sequential order of replanning guarantees that the conflict can be resolved. Provided that conflict resolution is not needed very often (i.e. there is enough free space available), the concurrent actions of the robots are only slightly degraded by the sequentiality of the replanning process. Most of the time the robots will be executing their path plans in parallel.

(b) Whenever a robot "perceives" any small obstacle during its locomotion it has to modify its motion on the fly. The reasoning module uses an obstacle avoidance ruleset that identifies the kinds of local obstacles that the robot might face and then uses key-frame motion control mechanism either to avoid the obstacle or to step on top of it. An example of a rule in such ruleset is:

> IF LOCAL OBSTACLE= PUMP AHEAD OF LEG_i AND PUMP CHARACTERISTICS ARE (STEEPNESS, FRICTION, SIZE, ETC.)
>
> THEN USE KEY-FRAME MOTION CONTROL MECHANISM TO MODIFY THE MOTION USING $MODIFIER_1$

The reasoning module associates two basic data structures with each robot: Motion Bit Vectors and Control Switches. Following [2] [12] the reasoning module associates a bit vector with every topological frame that exists in the system. A bit vector contains the state of the degrees of freedom that are currently affecting the associated robot's limb or object. Depending on the values of the control switches one or mixture of the motion control mechanisms is responsible for interpreting the goals, constraints, paths, directions (the details of the plan steps) as a series of binary vectors on the various topological frames.

For example if the control switches indicate that the key-frame motion control mechanism should be in control of a robot motion generation, then the bit vectors are interpreted this way: a bit is set in a motion bit vector of a particular limb when a continuous rotational/translational motion about/along the appropriate coordinate axis, is to be used to update the position of this limb in the next frame of motion. The key-frame motion control mechanism will access the semantic representation of the current action plan (see Figure 4) and identify the characteristics of the motion (its path, relative to, rate, until, etc.).

If the control switches indicate that the kinematics motion control mechanism should be in control of robot motion generation, then the bit vectors are interpreted this way: if a bit is set in a motion bit vector of a particular limb then a particular kinematic motion process can effect this limb. There is one bit for each kinematic motion process. Some of the kinematic and dynamic motion processes are rise, fall, jump, swing, hop, lean, pivot, flex a link, bend a link, turn, push, pull, release, grasp, etc.

In general there are two types of kinematic motion process: local kinematic motion processes that effect only the associated limb (i.e. sets bits in only the associated limb's bit vector- e.g. "flex a link"), and global kinematic motion processes that effect several limbs (i.e. sets bits in their bit vectors- e.g. "push"). In the dynamics motion control mechanism case the interpretation of the bit vectors is identical to the kinematics case except for having dynamics motion processes instead of kinematic ones.

The motion processes are executed on each iteration of the animation. Each degree of freedom in each limb is considered separately. At start of processing for a degree of freedom, its internal rotation (in case of kinematic motion control) or torque/force (in case of dynamic motion control) is set to zero. Then the state bit vector is examined to determine the motion processes that are to be executed. Each motion process uses the current state of the limb, plus its own parameters (stored with the state vector bit) to compute a contribution to the internal rotation (in case of kinematic motion control) or torque/force (in case of dynamic motion control). At the end of this process, new internal rotations or torques/forces will be generated for each limb of the associated robot. These internal rotations or torques/forces will represent the new inputs for the kinematic and dynamic motion control mechanisms.

More details about the motion processes and bit vectors in the particular case of dynamics motion control are treated elsewhere [2].

Similarly, the control switches can be set to indicate that any mixture of the previous motion control mechanisms are participating in the motion production. It is important to mention here that the motion control mechanisms for the articulated bodies (key-frame- kinematics- dynamics) are not mutually exclusive, in the sense that a mixture of kinematics and dynamics has been demonstrated successfully in [9] [10] [30], key-frame and kinematics in [21], key-frame and dynamics in [11]. One of the important features of the proposed model is that it is an open-ended model, in the sense that any new articulated body motion control mechanism could be incorporated into the model. One of the problems with our previous animation system [1] [2] is that the structures of the animated figures and their parameters were hard-coded into the animation routines. In this model, this is replaced by separate agent models that contain complete descriptions of the animated robots and their environment and separate motion control mechanisms that can work on them.

[3] EXPERIMENTAL SYSTEM

An experimental system is under development using an *IRIS** 2400 and several *SUN** workstations [13]. The system is divided into two main components, which are shown in figure 6. The first component, called the front end, is responsible for displaying the robot models and interacting with the animator (see figure 1). This component of the animation system resides on the IRIS and is responsible for displaying and controlling the different expert animator agent's motions. The second component associates a SUN workstation with each expert animator agent. Each agent controls the motion of a particular robot in the system.

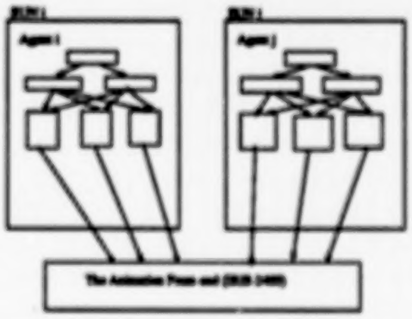


Figure 6:The Experimental System

The two components of the animation system (the IRIS and the various SUNs) communicate by sending packets over an interprocess communications facility (*Unix** sockets: since the workstations are connected by an ethernet). The front end invokes one of the backends when the animator wants to assign a task to the corresponding robot. Upon invocation, the expert animator agent of the robot reads the task description and calls the Task Planning and Execution Monitoring module to produce a plan for the locomotion task (In our experimental system we only restricted ourselves to locomotion issues, the manipulation capabilities of the robots were ignored for the purpose of simplifying the experimental system). The reasoning agent module will evaluate the context of the locomotion for the first plan step and will decides on the animation level of the locomotion production (key-frame, kinematic, dynamic, or any mixture) according to its sophistication-level ruleset.

---

* IRIS is a trademark of Silicon Graphics, Inc., SUN is a trademark of SUN Microsystems, Inc., and Unix is a trademark of AT&T Bell Laboratories.

The appropriate motion control mechanism will take over plan step execution and will send a set of packets to the front end. These packets represent the next animation frame to be displayed for the appropriate robot. There is one packet in this set for each limb of the robot, giving its current joint angles. There is also a packet specifying the current position of the body limb of the robot within the environment. At this point the front end responds with one or more packets. These packets are used to inform the expert animator agent of possible collisions with other robots, any small obstacles that are in the robot's way, and the current context of the robot's motion (e.g, the location of the camera with respect to the robot, whether the robot is out of sight in the next scene, etc.). The last packet in this exchange is a Next-frame packet sent from the front end to the expert animator agent. At this point the expert animator agent starts the next frame calculation cycle.

Similar packet exchange take place at the same time between the expert animator agents and the front end. These packet exchanges are synchronized by the front end to ensure that the front end and all the agents are always in step.

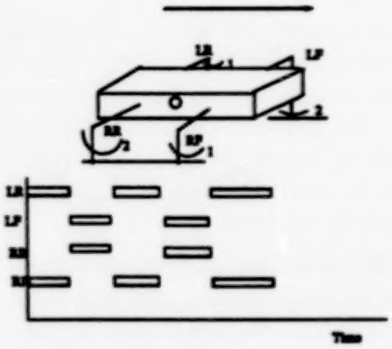


Figure 7:A Robot trotting

The object-oriented approach was quite suitable for the multi-robot animation environment. Viewing each expert animator agent as a distinct object facilitated their separate implementations on the various SUN workstations. In the same manner, the graphics module on the IRIS is also managed according to the object-oriented approach. The interprocess communication packets could be viewed as the messages that are exchanged among the various objects in the system.

The expert animator agents use three locomotion control mechanisms:

(1) key-frame locomotion control mechanism: This is the simplest mechanism to produce a robot's motion. The coordination and synchronization of the joint rotations of the legs are programmed via various locomotion scripts. For example a trotting script is shown in Figure 7.

(2) kinematic locomotion control mechanism: This was based on the work of Girard at Ohio State [9].

(3) dynamic locomotion control mechanism: This was based on our previous work on dynamic locomotion of the single articulated robot [13].

A simple planning algorithm based on A^* was implemented in each expert animator agent with a simple locomotion ruleset that describe the context for using each locomotion skill. An example of one such rule is:

$$IF\ LEG_1[lc_i(X)]\ \&\ LEG_2[lc_i(Y)]\&\ LEG_3[lc_i(Z)]\ \&\ LEG_4[lc_i(W)]\&\ NAVIGATION\ GOAL = A \rightarrow B$$

$$\&\ ROAD(A,B) = FLAT\&\ DISTANCE(A,B) \leq 5\&\ INITIAL\text{-}LOC(A) = NS$$

$$\& \ \mathit{FINAL{-}LOC}(B) = \mathit{NS} \ \ \mathit{THEN}$$

$$\mathit{ACTIVATE\ NC}\ [lc_1(X'), lc_1(Y'), lc_1(Z'), lc_1(W')]\ \mathit{UNTIL}\ X_E = X\textit{-coordinate of the final loc}(B)$$

$$\& Y_E = Y\textit{-coordinate of the final loc}(B)\ \mathit{WITH}\ \textit{linear speed} = [a, b]$$

$$\mathit{DELETE\ FROM}\ WM_1{:}\textit{old } LEG_i \textit{ locs}\ ,\mathit{ADD\ TO}\ WM_1{:}$$

$$\mathit{INITIAL{-}LOC}(B) = \mathit{NS}\ \ \& LEG_1\,[lc_1(\textit{final pos})] \& LEG_2\,[lc_1(\textit{final pos})]$$

$$\& LEG_3\,[lc_1(\textit{final pos})] \& LEG_4\,[lc_1(\textit{final pos})]$$

This represents a rule for the usage of the locomotive skill, walking. It provides the conditions under which this skill should be selected to implement the locomotion from A to B. In the action part, it describes how to derive the dynamics routines to implement the skill. Intuitively, this rule says: If the robot's four legs are in some orientation (X,Y,Z,W) within a particular leg cycle (i) (see Figure 1) ,and the goal broadcast by the navigator system is ($A \rightarrow B$) such that the road from A to B is flat and the distance between these two points is less than or equal to 5 units of distance, and both points are located on NS (North-South) direction; then start to drive the NC with the lc_1 leg cycles for all legs such that the legs start from the closest positions to the initial leg settings (X',Y',Z',W') in lc_1. This is in order to facilitate the smooth transitions between different types of gaits. The linear body speed is within the range [a,b], and the stopping conditions are of the destination point B (within some tolerance).

The screen layout for the front end is shown in Figure 1. The display screen is divided into six main sections, called the Terrain Display area, the Robot Selection area, the Locomotion Ruleset area, the Small obstacles Ruleset area, the User Interface area, and the command area. The Terrain Display area display the environment and the various robots at motion. The animator can change the viewing camera position and orientation using the mouse (e.g, zoom-in or zoom-out). As long as the cursor is in the Terrain Display area, the animations of the various robots are displayed. The animator can interrupt the animations by moving the cursor (using the mouse) to any area other than the Terrain Display area.

At the start of the system, the animator moves the cursor to any of the robots and select the robot using the mouse. The robot's name will show on the Robot Selection area of the screen, and the robot color will get changed to indicate that this robot is the current designated robot. At this point the animator can either define a mission for the designated robot, or investigate any enquires about its current status (see below). If the animator wants to assign a mission to the designated robot, s/he should move the cursor to the User Interface area and type in the task for the designated robot (e.g, Goto Place (x_i, z_i)). At this point, when the animator moves the cursor to the Terrain Display area the robot's expert animator agent (on the appropriate SUN) will take over the execution and animation of the task.

At any point the animator can use the Command area to ask a specific robot to move faster; slower; to change the projection view (e.g, orthogonal, prospective); redefine a mission for a particular robot, etc. The Locomotion Ruleset area, the Small Obstacle area, and the User interface area will reflect the current status of any selected robot. The animator can move the cursor and selects any of the robots during the animation. In this case all the internal information of this robot's expert animator agent is available to the animator to manipulate. The status of the designated expert animator agent will be reflected in the various screen areas: (1) The Locomotion Ruleset area will display the current rules that are producing the locomotions of the designated robot. (2) The Small Obstacles Ruleset area will continuously display the current rules that the designated robot's expert agent is using to avoid or overcome the various small obstacles that it is facing as the animation progresses. (3) The User Interface area will display the following information based on the animator requests: (a) the locations of the support legs, (b) the polygon of stability, (c) the predicted polygon of stability when the swinging legs are lowered; (d) the location of the center of gravity, (e) the reachable areas of the legs, (f) the answers to any Reasoning enquiries submitted by the animator.

In the experimental system the sequential ordering of path replanning of section 2.3 was implemented, but no animator explanation (reasoning) capabilities were implemented.

[4] CONCLUSION

The paper describes an advanced expert animator model for animating a group of articulated robots in a three-dimensional environment. The model shifts the burden of human animator involvement with the mechanisms of the robots' motion to various programmed expert animator agents. These agents have computational understanding of motion and its semantics in a way that each handles its robot's behaviour autonomously and communicates with other agents as well as the animator. The design principle of the agents is based on knowledge engineering methods (object-oriented and rule-oriented programming) integrated with computer graphics. The potential of our model is shown by a simple experimental system that was limited to locomotion activities.

REFERENCES

[1] W. Armstrong ,M. Green ,The Dynamics of Articulated Rigid Bodies for Purpose of Animation ,Proceedings of Graphics Interface'85 ,1985

[2] W. Armstrong ,M. Green ,R. Lake Near-Real Time Control of Human Figures Models ,Graphics Interface-86 ,1986

[3] A. Bertoni, G. Haus, G. Mauri, and M. Torelli, "A Mathematical Model for Analyzing and Structuring Musical Texts, Interface 7, 1978.

[4] T. Calvert, J. Chapman, and J. Lindis, "Notation of Dance with Computer Assistance", in New Directions in Dance, D. Taplin, ed., Pergamon Press, Toronto, 1979.

[5] A. Camurri, P. Morasso, V. Tagliasco, and R. Zaccaria, "Dance and Movement Notations", in Human Movement Understanding, North-Holland, 1986.

[6] R. Davis, "Report on the Workshop on Distributed AI", SIGART Newsletter, no. 73, Oct. 1980.

[7] R. Davis, "Report on the Second Workshop on Distributed AI", SIGART Newsletter, no. 80, April 1982.

[8] M. Fehling and L. Erman, "Report on the Third Annual Workshop on Distributed Artificial Intelligence", SIGART Newsletter, no. 84, April 1983.

[9] M. Girard and A. Maciejewski, " Computational Modeling for the Computer Animation of Legged Figures", Computer Graphics 19,3, 1985.

[10] P. Isaacs, M. Cohen, "Controlling Dynamic Simulation with Kinematic Constraints, Behaviour Functions and Inverse Dynamics", Computer Graphics, v. 21, no. 4, July 1987.

[11] D. Lundin, "Simulation", Advanced Computer Animation Tutorial, Siggraph 1986.

[12] T. McMahon, "Mechanics of Locomotion", the International Journal of Robotics Research, 1984, also T. McMahon: Muscles, Reflexes, and Locomotion, Princeton University Press, Princeton N.J., 1984.

[13] A. Mohamed ,A Learning Apprentice System For Locomotion Control of Articulated Bodies ,A Ph.D. thesis in preparation. ,1988

[14] A. Mohamed, W. Armstrong, An Experimental Autonomous Articulated Robot That Can Learn, to appear in the proceeding of the 3rd International Conference on CAD/CAM Robotics & Factories of the Future, Southfield Michigan, August 14-17, 1988.

[15] A. Mohamed, W. Armstrong, A Hybrid Numerical/Knowledge-Based Locomotion Control System for a Multi-legged Manipulator Robot, to appear in the proceeding of the 8th International Workshop on Expert Systems and their Applications, to be held in Avignon, France, June 1-5, 1988.

[16] P. Morasso, V. Tagliasco, "Advances in Psychology: Human Movement Understanding", North-Holland, 1986.

[17] F. Moore, "Introduction to Music Synthesis Using CMUSIC", Technical Report University of San Diego, California, 1982.

[18] Z. Pylyshin, Computation and Cognition, MIT Press, Cambridge, 1984.

[19] D. Robson, "Object-Oriented Software Systems", BYTE, V.6, no.8, 74/86, 1981.

[20] E. Rich, Artificial Intelligence, McGraw-Hill, Reading, 1983.

[21] G.Ridsdale, S. Hewitt, and T. Calvert, "The Interactive Specification of Human Animation", Proc. Graphics Interface '86, Vancouver, 1986.

[22] R. Ryman and B.Singh, "the Benesh Notation Computerized Editor", Proc. Dance in Canada Conf., June 1982.

[23] R. Schank, "Identification of Conceptualizations underlying Natural language", In R.C. Schank & K.M. Colby (Eds), Computer Models of thought and language, San Francisco: Freeman, 1973.

[24] G. Savage and J. Officer, "Choreo: An Interactive Computer Model for Choreography", Proc. of the 5th Man-Machine Comm. Conf., Calgary, Alta, 1977.

[25] M. Stefik, et al, "Rule-Oriented Programming in LOOPS", Symposium on knowledge Engineering and Artificial Intelligence, Information Processing Society of Japan, 65/71, 1985.

[26] R. Smith, "Report on the 1984 Distributed Artificial Intelligence Workshop", AI Magazine, v.6, no.3, Fall 1985.

[27] in Planning, Geometry, and Complexity of Robot Motion, J. Schwatrz, M. Sharir, J. Hopcraft (Eds), Ablex Publishing Corporation, 1987.

[28] M. Turvey, C. Fowler, "Skill Acquisition" An Event Approach with Special Reference to Searching for the Optimum of a function of Many Variables", in G.E. Stelmach (Ed.), Information Processing in Motor Control and Learning, New York, Academic Press, 1978.

[29] N. Winkless and I. Browning, Robots on Your Doorsteps, Robotic Press, Portland, OR, 1978.

[30] J. Wilhelms ,B. Barsky ,Using Dynamic Analysis for the Animation of Articulated Bodies such as Human and Robots ,Proceedings of Graphics Interface'85 ,1985

N89-15590

Graphic Simulation Test Bed for Robotics Applications in a Workstation Environment

J. Springfield, A. Mutammara, G. Karsai
G.E. Cook and J. Sztipanovits
Department of Electrical Engineering
Vanderbilt University, Nashville, TN 37235
K. Fernandez, Ph.D.
Automation and Robotics Research,
Marshall Space Flight Center, Huntsville, AL 35812

Abstract -- Graphical simulation is a cost-effective solution for developing and testing robots and their control systems. The availability of various high-performance workstations makes these systems feasible in everyday practice. Simulation offers preliminary testing of systems before their actual realizations, and it provides a framework for developing new control and planning algorithms. On the other hand, these simulation systems have to have the capability of incorporating various knowledge-based system components, e.g. task planners, representation formalisms, etc. They also should have an appropriate user interface, which makes possible the creation and control of simulation models.

ROBOSIM was developed jointly by MSFC and Vanderbilt University, first in a VAX environment. Recently, the system has been ported to an HP-9000 workstation equipped with an SRX graphics accelerator. The user interface of the system now contains a menu- and icon-based facility, as well as the original ROBOSIM language. The system is also coupled to a symbolic computing system based on Common Lisp, where knowledge-based functionalities are implemented. The knowledge-based layer uses various representation and reasoning facilities for programming and testing the control systems of robots.

Introduction

Robots are becoming increasingly important in every area of industry. Along with an increase in robot sophistication and payload capability there is an increase in the possibility of damage to the robot and to the workcell, especially during the development of the system and the algorithms or programs that will drive the robot. Simulation is even more important for robotic systems designed to operate in space. These robots, which are desgned to operate in zero-g, can not be tested in full on the ground. While robot simulation programs have existed for a while, the advent of high-speed graphics workstations allows real-time graphical simulation at a fraction of the cost as in the past.

ROBOSIM [1,2] was developed jointly between NASA and Vanderbilt University. ROBOSIM has been running on a DEC VAX 11/780 with the capability to use terminals with TEK4014 graphics compatibility. Interfaces for Evans & Sutherland PS3xx, GTI Poly 2000, and Silicon Grahphics IRIS are supported also.

ROBOSIM operates via an interpreted program that the user writes. The program consists of commands that create various solid and planar primitives that can be rotated and translated by other commands. In this way, the links of the robot are built up, with the relationships between the links following the Denavit-Hartenberg convention. ROBOSIM generates structures describing the physical structure, its kinematics, and its mass properties. With this information everything is known except for joint position, velocity, and torque limits, which are specified in the actual simulation. Using this information, all aspects of the simulation can be implemented, while reducing the possibility of data skewing, as the physical model, the kinematics, and the mass properties are calculated at the same time from the same program. All data is provided for collision detection, graphics display, and dynamics.

This full implementation of ROBOSIM has been ported to a Hewlett-Packard 350SRX graphics workstation. Several additional features have been added to exploit the capabilities of this workstation, such as the 3D graphics editor.

Graphics Editor

The capabilities of the workstation allow a much friendlier user-interface. X-Windows and the 3D graphics library (along with the special-purpose graphics hardware) provide the basis for a sophisticated means of designing robots. The editor utilizes menus and a mouse to provide simple methods to use the system. Also, a "box" of analog knobs allow the viewpoint to be changed, colors altered, and various other parameters concerning the graphic display to be set as desired. Although similar to some CAD systems, the editor is designed with ROBOSIM in mind.

The editor uses object oriented methods in its operation. Different types of objects can be created, such as boxes, cylinders, and custom designed objects. The reference frames of the links are also defined as objects. After creation, objects can be rotated, translated, deleted, and resized. Also, the objects can be attached together. This attachment is hierarchical in that there is a parent object and a child object. Any rotation or translation on a parent object propogates to any child object and its children as well. Once objects have been attached then a resizing operation on either will result in the relationship between them remaining constant. In this way complex, custom-designed, yet generic objects can be created. All that is necessary is to specify the dimensions of the components. This attachment allows more complex objects to be constructed without losing the ability to operate on the primitives.

Once the robot link has been built up, the structures can be saved for later editing. However, this editor will also generate a ROBOSIM program. The editor goes through the hierarchy, propogating every operation down. This yields a primitive object and a set of translations and rotations. Once this is achieved it is a fairly straigtforward procedure to automatically generate the ROBOSIM code. In this way the editor does not need to concern itself with mass properties or kinematics, as this is automatically produced by the ROBOSIM object compiler.

Object Compiler

The object compiler takes the output of the graphical editor and generates all of the data about a link: the polygonal model, the kinematic model, and the inertial data. Figure 1 shows a session during the editor in which a link is built. Figure 2 is a listing of the ROBOSIM code generated by the editor. And Figure 3 is an outline of the

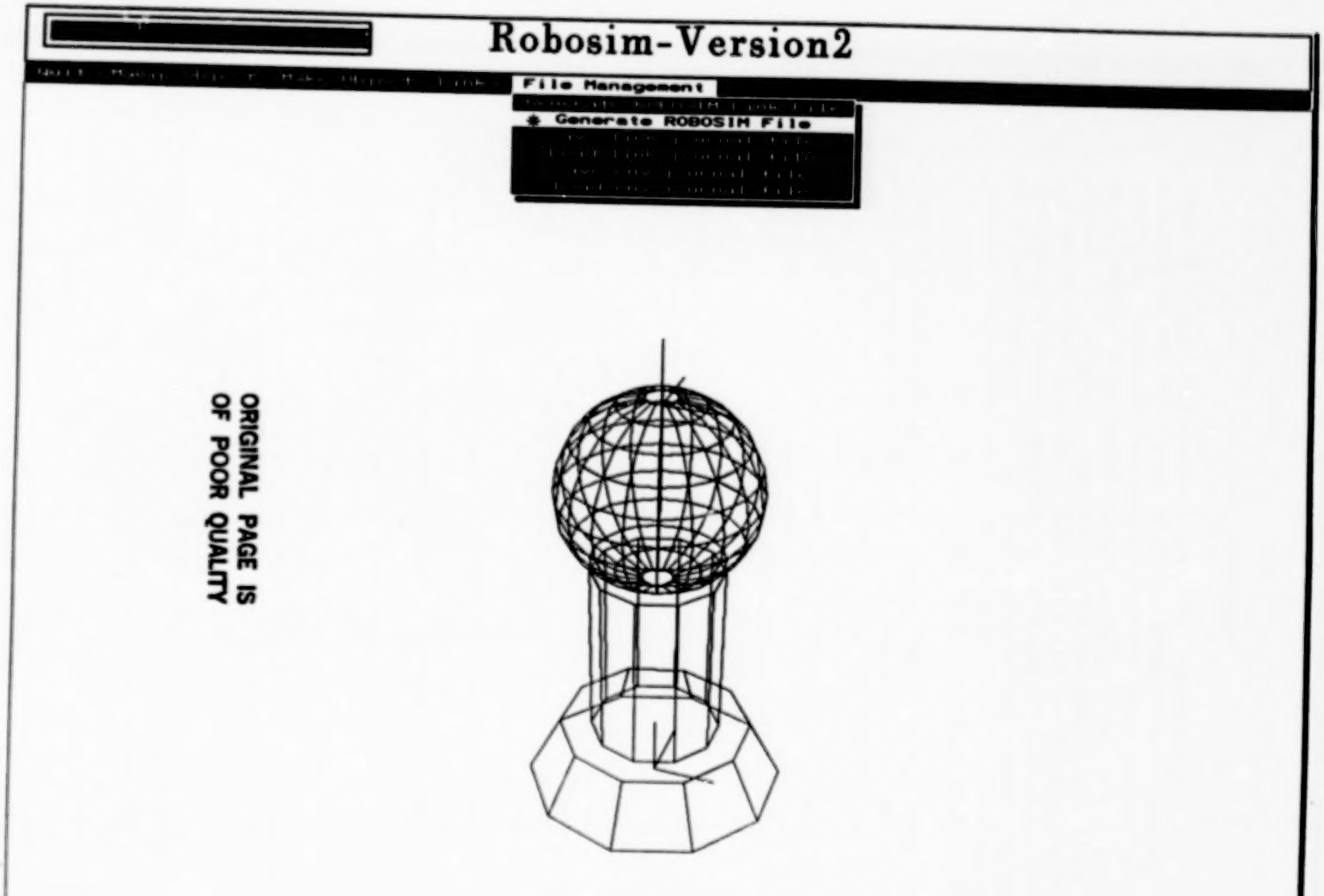


Figure 1. 3D Graphical Editor

ORIGINAL PAGE IS
OF POOR QUALITY

```
LOOK-FROM X=-150., Y=50., Z=100.
LOOK-AT X=0., Y=0., Z=0.
CLEAR
STORE B
F-JOINT-I
      TRANSLATE X=0.000, Y=0.000, Z=-43.333
      ADD B
      STORE B
CLEAR
R-JOINT-I+1
      TRANSLATE X=0.000, Y=0.000, Z=23.811
      ADD B
      STORE B
CLEAR
SPHERE R=15.000
      TRANSLATE X=0.000, Y=0.000, Z=10.239
      ADD B
      STORE B
CLEAR
CYLINDER R=10.000, H=30.000
      TRANSLATE X=0.000, Y=0.000, Z=-18.333
      ADD B
      STORE B
CLEAR
TRUNCATED-CONE RL=20.000, RU=15.000, H=10.000
      TRANSLATE X=0.000, Y=0.000, Z=-40.000
      ADD B
      STORE B
CLEAR
LOAD B
STORE-LINK ROBOT.LOC
VIEW
END
```

Figure 2. ROBOSIM code generated by editor

<table>
<tr><th>Row</th><th>Col 1</th><th>Col 2</th><th>Col 3</th><th>Col 4</th></tr>
<tr><td>1</td><td>THETA</td><td>DZ</td><td>DA</td><td>ALPHA</td></tr>
<tr><td>2</td><td>JA1</td><td>JA2</td><td>JTYPE1</td><td>JTYPE2</td></tr>
<tr><td>3</td><td colspan="4">AINERT (4X4)</td></tr>
<tr><td>7</td><td colspan="4">AJNT-I (4X4)</td></tr>
<tr><td>11</td><td colspan="4">AJNT-I+1 (4X4)</td></tr>
<tr><td>15</td><td colspan="4">AMAT (4X4)</td></tr>
<tr><td>19</td><td>NVEC</td><td>UNUSED</td><td>UNUSED</td><td>UNUSED</td></tr>
<tr><td>20</td><td>X1</td><td>Y1</td><td>Z1</td><td>D1</td></tr>
<tr><td>21</td><td>X2</td><td>Y2</td><td>Z2</td><td>D2</td></tr>
<tr><td>:</td><td>:</td><td>:</td><td>:</td><td>:</td></tr>
<tr><td>NVEC+19</td><td>XNVEC</td><td>YNVEC</td><td>ZNVEC</td><td>DNVEC</td></tr>
</table>

Variable Definitions:

THETA = Denavit-Hartenberg parameter
DZ = Denavit-Hartenberg parameter
DA = Denavit-Hartenberg parameter
ALPHA = Denavit-Hartenberg parameter
JA1,JA2 = joint defined flag
JTYPE-I,I+1 = joint type -> Revolute,Prismatic,Fixed
AINERT = generalized link inertia
AJNT-I,I+1 = transforms of input and output frames
AMAT = link's A-matrix
NVEC = number of vectors in list
Xi,Yi,Zi = x,y, and z component of vector
Di = move or draw vector

Figure 3. Data Structure of Link

data structure generated by ROBOSIM.

Simulator

The basis of the simulator is a library of C routines. These routines operate directly on the ROBOSIM structures. This allows the specifics to be hidden from the user, unless direct manipulation is required by the simulation. With these routines, one can implement fast, very specific simulations that are written in C and linked with the simulation library. However, one can also implement an interactive, general-purpose simulation in those cases where real-time simulation is not required.

These routines allow various robots, environments, and objects to be loaded separately and combined in any configuration. They allow commands similar to many robot programming languages to drive the robots. Collision detection can be turned on or off, and forces and torques can be returned to the simulation or used for control of the robot. Straight-line motion is included as well as a facility to move along any parametrically defined function. The joint angles can also be returned to the simulation for its use.

Although not included at this time, translators for many robot programming languages will be available in order to download developed programs directly to the physical robot.

Knowledge Representation Facilities

It is important to look at the ideas presented above, along with areas of future expansion, from the point of view of knowledge representation. Different choices in representation can result in changes in efficiency, flexibility, system requirements, and user-friendliness.

The techniques of Knowledge Representation (KR, for short) have been developed in the framework of Artificial Intelligence research. For engineering purposes KR techniques offer enhanced capabilities for modeling, where in addition to traditional numerical models, various symbolic models can also be used. The latter might include various declarative languages which represent technical concepts and entities [3].

This idea is applied in the robot simulation system as follows. Robot modelling and programming are supported by various declarative languages, which supply a knowledge-based description of (1) the robot, (2) its control system, (3) its environment, and (4) its task to be performed. Currently, these declarative languages are realized in a Lisp framework.

The geometric objects in robot modelling together constitute a graphic model of a robot or objects from its environment, and they can be used for various other considerations dependent on the objects, e.g. collision detection, dynamics, and task planning. For this application a declarative language has been defined, which supports the hierarchical representation of geometrical objects by using object--oriented programming techniques, as discussed previously.

The computations for the forward and inverse dynamics can be synthesized from the geometrical and physical model. This happens as follows: from the declarations describing the geometry and the physical properties a dataflow graph is synthesized, which represents the flow of computations needed to solve the dynamics problem. This graph is executed on the Multigraph Architecture (MA) [4,5], which makes possible the integration of symbolic and numeric computing in such a way, that the structure of a complex computation is represented in a declarative framework, while the computational primitives can be represented in terms of fast and efficient numerical algorithms.

Joint constraints and geometrical properties together are used for collision detection, where the detection algorithm also utilizes the representation schemes mentioned above.

The simulation of a robot system generates numerical and graphical output: numerical output contains information about joint angles, forces and torques, etc., while graphical output provides visual feedback for the designer. To control a robot which has been modelled using the facilities described above, one can create robot controller structures using a similar knowledge representation scheme. Here, again, a declarative language is used, which lets the designer describe a control system in terms of signal processing blocks, which implement various functionalities of the control system. This declarative language is also supported by a graphical editing technique, which uses icons to specify the blocks, and they can be connected to form a signal processing graph. The resulting declarations are interpreted to generate the running signal processing system using the Multigraph Architecture.

The tasks to be performed are represented in declarative terms also. Using hierarchical decomposition, robot control sequences are represented in symbolic form, from which a task planner synthesizes the actual sequence of control actions needed to accomplish a certain goal. The hierarchical planner generates a chain of objects, which represent the steps to be performed. Each such step is linked to a low-level control scheme which implements that step. Now if the step has successfully been executed (i.e. the low-level control scheme has not signalled an error) the execution proceeds, while if there were a problem, the execution (1) might go along either a new path, or (2) a completely new sequence is synthesized together with its low-level control blocks. This dynamic replanning might influence the computational model of control, dynamically changing the structure of the control system. One example might be as follows: if the collision detection scheme signals an error to the controller system, that event might initiate the restarting of the task planner to re-generate or structurally modify the step sequence and/or its associated controller. This flexible way of representing tasks, task steps and low-level controllers was made possible by using the integrated computational model of the MA.

Conclusions

The need for competent, comprehensive robot simulators has been well established. In order to provide adequate simulation capability, the system must allow graphical, as well as dynamic, simulation. The new generation of graphics workstations provide for extremely fast graphics output, while freeing the main processor for kinematic and dynamic calculations. However, mere number crunching is not enough to create a better simulator. Symbolic planners and other such AI programs are necessary in order to fully meet the demanding requirements that are being asked for.

References

1. Fernandez, K.R. and Cook, G.E., "Use of Computer Graphic Simulation Techniques for Robot Control System Development", IEEE Computer Society, Proc. 18th Southestern Symposium on System Theory, Knoxville, TN, April 7-8, 1986, pp.433-441.

2. Fernandez, K.R., "The Use of Computer Graphic Simulation in the Development of Robotic Systems", Acta Astronautica,Vol. 17, No.1, pp.115-122, 1988.

3. Karsai, G., "Declarative Programming Techniques for Engineering Problems", Ph.D. Thesis, Vanderbilt University, August 1988.

4. Sztipanovits, J., "Execution Environment for Intelligent Real-time Control Systems," Proc. of the NASA/JPL Symposium on Telerobotics, 24 pgs., Pasadena, CA, 1987.

5. Biegl, C.A., "Design and Implementation of an Execution Environment for Knowledge-Based Systems", Ph.D. Thesis, Vanderbilt University, December 1988.

DESIGN OF A SIMULATION ENVIRONMENT FOR LABORATORY MANAGEMENT BY ROBOT ORGANIZATIONS *

Bernard P. Zeigler, François E. Cellier, Jerzy W. Rozenblit
Dept. of Electrical and Computer Engineering
University of Arizona, Tucson, AZ 85721

ABSTRACT

This paper describes the basic concepts needed for a simulation environment capable of supporting the design of robot organizations for managing chemical, or similar, laboratories on the planned U.S. Space Station. The environment should facilitate a thorough study of the problems to be encountered in assigning the responsibility of managing a non-life-critical, but mission valuable, process to an organized group of robots. In the first phase of the work, we seek to employ the simulation environment to develop robot cognitive systems and strategies for effective multi-robot management of chemical experiments. Later phases will explore human-robot interaction and development of robot autonomy.

INTRODUCTION

This paper describes the design of a simulation environment capable of supporting the study of robot organizations for managing chemical, or similar, laboratories aboard Space Station. Laboratory management includes the servicing and calibration of equipment, the set-up of experiments to external specifications, the monitoring and control of experiments in progress, the measurement of results, and finally the recording and analyzing of data. The environment should facilitate a thorough study of the problems to be encountered in assigning the responsibility of managing a non-life-critical, but mission valuable, process to an organized group of robots.

Our ultimate research goals are to employ the simulation environment to develop robot cognitive systems and strategies for effective multi-robot management of laboratory experiments. We seek an understanding of how to partition automation tasks between hard and soft forms, i.e., between "intelligent" instruments and flexible robots. We shall assess the nature of human supervision initially required, and seek to develop workable man-robot co-operation protocols. Also, we seek to develop robot learning paradigms such that the autonomy of a robotic organization increases with experience, and consequently, the need for human supervision and intervention is diminished.

It is timely to begin exploration of advanced robot-controlled instrumentation. For example, handling fluids in orbit will be essential to many of the experiments being planned in manufacturing and biotechnology. However, the microgravity conditions of space necessitate radically different approaches to fluid handling than common on earth. As experience in space accumulates, approaches and instrumentation will likely undergo continual modification, enhancement, and replacement. Thus, robots for managing such equipment must be sufficiently intelligent and flexible so that constantly changing environments can be accommodated. Given its importance and novelty, we have chosen fluid handling in microgravity as the focus for our laboratory environment.

Discrete event simulation and AI knowledge representation schemes form a powerful combination, called knowledge-based simulation, for studying intelligent systems in a realistic manner [1,3,5,6,7,8]. DEVS-Scheme [14,15] is a knowledge-based simulation environment for modelling and design that facilitates construction of families of hierarchical models in a form easily reusable by retrieval from a model base. The laboratory environment, implemented in DEVS-Scheme, is being constructed on the basis of object-oriented and hierarchical models of laboratory components at multiple levels of abstraction.

---

* Supported by NASA-Ames Cooperative Agreement No. NCC 2-525, "A Simulation Environment for Laboratory Management by Robot Organizations"

The robot cognition model is based on an "action-by-exception" principle control of a knowledge base of Model-Plan Units (MPUs). Davis [2] described a similar knowledge based simulation environment in which agents are governed by a script (plan of actions) and a set of production rules for deciding when to proceed from one phase of the plan to the next, which detailed actions to execute within a phase, and what to do if one plan has to be replaced by another one. Holland's [4] classifier (parallel production rule) system provides concepts for sequencing robot actions.

In designing the robot models, we assume that necessary mobility, manipulative and sensory capabilities exist so that we can focus on task-related cognitive requirements. Such capacities, the focus of much current robot research, are treated at a high level of abstraction obviating the need to solve current technological problems. Organizational issues are introduced from the beginning since individual robot capabilities may be much influenced by co-operative requirements.

A primary goal in the robot model design is to minimize the number of sensory inputs that the system must attend to at any one time. Except at critical selection points, attention is focused on only those aspects of the environment dictated by the currently activated MPU. While an MPU behavior lies within its envelope, no other MPU can supplant it, even if its initialization conditions better fit the current situation. One of the primary goals of the project will be to judge whether these principles provide a workable basis for intelligent robot design. For example, such robots may be so single-minded as to be incapable of flexibly responding to an unknown or changing environment.

DEVS-Scheme Simulation Environment

DEVS-Scheme [14,15] is a knowledge-based simulation environment for modelling and design that facilitates the construction of families of models in a form easily reusable by retrieval from a model base. The environment supports the construction of hierarchical discrete event models, and is written in the PC-Scheme language which runs on IBM compatible microcomputers and on the Texas Instruments Explorer System.

System Entity Structure Knowledge Representation

Model specification and retrieval in the DEVS-Scheme simulation environment is mediated by a knowledge representation component designed using the system entity structuring concepts [16,19]. The system entity structure incorporates decomposition, taxonomic, and coupling knowledge concerning a domain of real systems [10,12]. A user prunes the entity structure according to the objectives of the modelling study obtaining a reduced structure that specifies a hierarchical discrete event model [11]. Upon invoking the transform procedure, the system searches the model base for components specified in the pruned entity structure, and synthesizes the desired model by coupling them together in a hierarchical manner. The result is a simulation model expressed in DEVS-Scheme which is ready to be executed to perform simulation studies.

Process Laboratory Model

The laboratory environment is being constructed on the basis of object-oriented and hierarchical models of laboratory components within DEVS-Scheme. Laboratory configurations will be determined by pruning the entity structure knowledge representation. The laboratory model is being designed to be as generic as possible. However, as stated, the focus will be upon fluid handling in microgravity which presents a variety of problems that are unique to space.

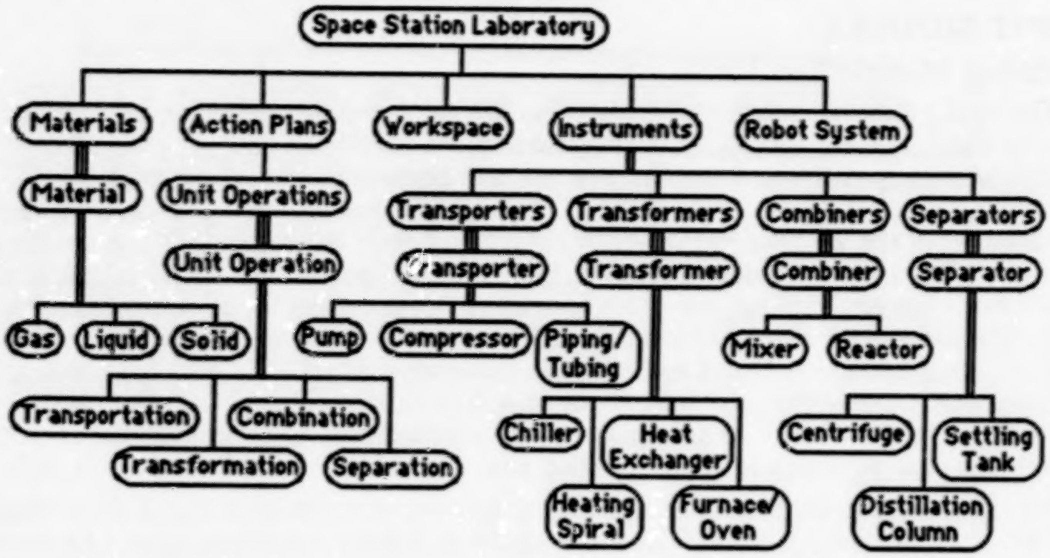


Figure 1. System Entity Structure for Space Station Laboratory

Figure 1 illustrates the approach taken. The entity structure for SPACE STATION LABORATORY decomposes this entity into MATERIALS, ACTION PLANS, a WORKSPACE, INSTRUMENTS, and the ROBOT SYSTEM. Each of the latter entities will have one or more classes of objects (models) expressed in DEVS-Scheme to realize it. MATERIALS will be specialized by physical state into the classes GAS, LIQUID, and SOLID, and will be further subclassified as needed. ACTION PLANS are composed of UNIT OPERATIONS which decompose into four types: TRANSPORTATION (which changes the physical co-ordinates of a material, e.g. pumping), TRANSFORMATION (which transforms the state of a single material), COMBINATION (which produces a new material from several input materials, e.g. chemical reactions), and SEPARATION (which partitions a material into several components).

UNIT OPERATIONS are carried out with one or more INSTRUMENTS, which may be TRANSPORTERS, TRANSFORMERS (e.g. centrifuges), COMBINERS (e.g. mixers), or SEPARATORS. To illustrate the special nature of space, consider transporters. Since air/liquid interfaces are not permitted under microgravity conditions, standard earth-bound containers, such as beakers, cannot be used. A design of a space adapted "beaker" would have an aluminium bottle containing an inflatable bag, which is the actual liquid container; liquid is injected/extracted by means of syringes; air pressure between the outside of the bag and the inside of the bottle wall ensures that the bag remains "full" at all times.

ACTION PLANS are sequences of UNIT OPERATIONS with associated MATERIALS and INSTRUMENTS. For example, injecting several liquids into a bottle, placing the bottle in a shaker, and then placing it in a heating spiral is a sequence related to experimentation with chemical reactions. Action plans have associated models whose construction will be discussed in the context of the robot Model-Plan Unit (MPU).

INSTRUMENTS have attributes which include operational conditions so that normal and abnormal operating behavior can be studied.

To set up a particular laboratory environment, the LABORATORY entity structure is pruned to create a pruned entity structure which will transform into a laboratory model. Constraints on the possible configurations of components, especially those imposed by microgravity and space environments, are captured by appropriate synthesis and selection rules [9].

ROBOT MODELS

Designing Model-Plan Units

The first stage in designing Model-Plan Units, involves modelling of continuous processes by discrete event models [13]. We start with a particular real process, such as heating liquid in a doubly-contained bottle. We identify regions of operation such as: "there is a sufficient amount of liquid in the bottle", "the liquid has reached the desired operational temperature", "the air pressure is too high", "the bottle has exploded". A continuous dynamical model is then developed for each region based on physico-chemical considerations. Boundaries between regions are then identified, and a discrete event model is specified whose internal events represent such transitions from one boundary to another. Scheduling of such transitions is based on time-to-next-event values obtained from trajectories of the dynamical model. For example, if the initial quantity of liquid and the rate at which it heats up are known as well as the increase in air pressure with temperature, then the time to reach the "air pressure too high" region can be pre-determined, and hence scheduled.

For each action on the real process, a normal state trajectory is identified in the continuous model and projected into the space of sensor measurements. An envelope is determined to enclose this projected state trajectory. This envelope specifies the variation to be tolerated in sensor measurements while still accepting an observed trajectory as normal. For computational feasibility, sensors with binary states (or a small set of discrete states) will be preferred, circumstances permitting. For example for sterilizing a liquid, that liquid should be heated up to at least 70 degrees centigrade, and should be kept at that temperature for a prescribed period of time. For other reasons, it may not be advisable to heat the liquid beyond 80 degrees centigrade. So, we may employ a sensor whose binary output indicates the temperature lying within, or outside of, the range 70-80. In addition to sensory boundaries, we employ timing information to determine normal operation. From estimated uncertainties in the initial state and parameters, the continuous model yields the window in which the time-to-next-event must lie for each state transition [18].

The plan of an MPU specifies a sequence of UNIT-OPERATIONS to be carried out to bring the real process from an initial state to a desired state. For example, an MPU for sterilizing water might specify filling a bottle with water, placing it in a heating spiral, and removing it when the required temperature of 70 degrees centigrade has been reached. Associated with each UNIT OPERATION is a set of sensors for detecting its initialization and goal states together with the time window in which each transition time must lie. If, for example, the time for the temperature sensor to change to its high state is not within that allowed, the MPU is disabled; if the time is within bounds, the next action, removing the bottle, is carried out.

Plan Abortion and Restart

Following disablement of an MPU, other MPUs may be activatable in the prevailing state. MPUs may, for example, exist if the process state is still a normal one; as a special case, the last activated MPU may be still have its initialization conditions satisfied. Consider for example, a situation where the envelope timing bounds associated with the action "heat water" were exceeded because the power to the heating spiral was turned off. The Selector must prevent the aborted MPU from gaining activation and attempting indefinitely to use an inoperative heater. This might be done with a recency, or frequency-of-use component in the selector's conflict resolution method.

Diagnostics are associated with MPU plan abortion. These diagnostics will attempt to discover faults which can be corrected to return the state to one in which normal operation can be resumed. Such diagnostics are guided by the sensor envelope and time window violations which caused the MPU to abort. For example in the above situation, given that the expected heating time was exceeded, a diagnostic may deduce that the heating spiral is not producing heat, and that one cause might be that power to it is turned off. The model underlying an MPU provides the basis for designing such diagnostic units.

Naturally, we expect that many unforeseen situations will emerge in simulation runs. In such cases, the robot system will fail. Since the current state of the process, as viewed in the last activated MPU and its sensor readings (or a record of the most recently activated MPUs) is available, we will be in a position to analyze what went wrong. It is expected that, in a real space laboratory, such events will also occur. Protocols will be investigated to alert human supervisors to such events, and facilitate restoring robot system operation.

IMPLEMENTATION

A simplified entity structure for the implemented robot organization is shown in Figure 2. This structure is transformed into a hierarchical model containing "controlled-models" at two levels. "Controlled-models" is a class in DEVS-Scheme which facilitates the construction of models containing arbitrary numbers of components which communicate with each other and with the outside world via a controller [17]. Thus at the top level, the ROBOT-SYSTEM is a controlled-model containing a SPACE-MANAGER as controller, and ROBOTS as components. Each ROBOT contains a motion, a sensory, and a cognition sub-system. The COGNITION-SYSTEM is itself a controlled model containing the SELECTOR as controller, and MPUS as components.

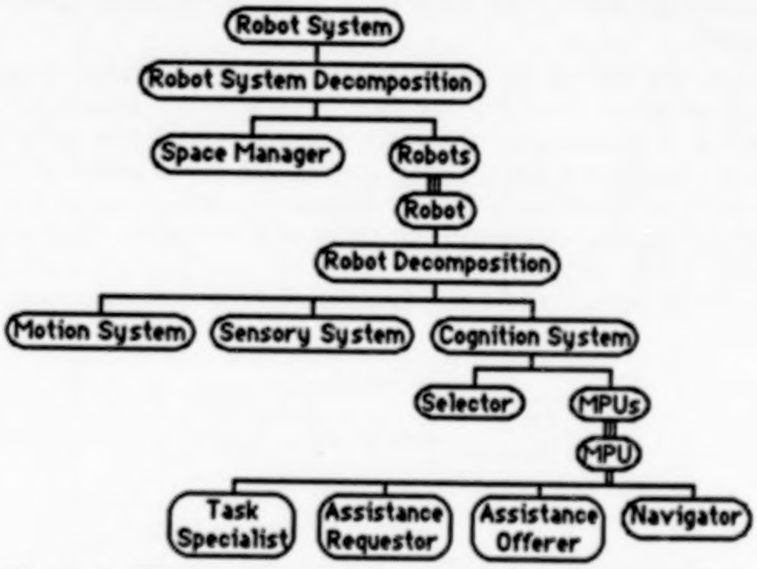


Figure 2. Entity structure for the Robot Organization

ROBOTS use their sensory subsystems to communicate with each other via the SPACE-MANAGER. When a ROBOT changes its position, its motion sub-system sends its new location to the SPACE-MANAGER which keeps track of the ROBOTS positions. When a ROBOT wishes to communicate with other ROBOTS, it sends its message to the SPACE-MANAGER which relays the message only to those ROBOTS that are located within the range of the sender. This range may vary depending on the channel on which the message is sent. In this way, different transmission media and sensory modalities may be modelled: light and vision, sound and hearing, pressure and touch, etc. Latency in message transmission, implemented by a delay in the SPACE-MANAGER, may also depend on the medium. Messages on certain channels, such as touch, are reflected back to the SPACE-MANAGER by sensory-subsystems upon arrival to a sensory subsystem, as well as being transmitted to the cognition system. Such echo messages are used by the original sender to ascertain relationships to the receiving robot.

Since the SPACE-MANAGER has complete knowledge of locations, it can detect collisions between ROBOTS. Space may be treated as a resource shared by its occupants so that collisions represent attempts to occupy the same space twice at the same time. The "management" of the resource is "dumb" if collisions are only *detected*. However, the SPACE-MANAGER may be given additional intelligence to co-ordinate the ROBOTS, e.g. to *prevent* collisions, thus modelling an artificial layer of supervision above the naturalistic one.

Within each ROBOT's cognition system, action-by-exception control ensures that an MPU, once initiated, retains activation until its plan is successfully executed, or until a significant discrepancy arises between the actual results of carrying out the plan and the results expected by the model. The SELECTOR is essentially a bi-state device whose state is determined by the MPU responses. In the closed state, it passes on the incoming sensory inputs to the activated MPU. Upon completion of the activated plan or upon receiving a discrepancy alert, it switches to the open state in which MPUs may vie for activation. Incoming sensory input is broadcast to all MPUs. The first MPU to respond to the input is established as the activated MPU. Once an MPU activation has occurred, the closed state is resumed.

As stated, a primary goal in this design was to minimize the number of sensory inputs that the system must attend to at any one time. This is achieved here by the fact that, in the closed state, the SELECTOR acts like a closed wire which uncritically transmits all inputs to the activated MPU. The latter only pays attention to those inputs which matter to achieve its goals.

In the future, we intend to implement more general conflict resolution schemes to determine which of the activatable MPUs will be granted permission to activate. MPUs will be arranged in a generalization hierarchy, an inverted tree with relatively few highly general MPUs at the lowest level. Such generalists cover most of the environment, but with less than fully efficient capabilities. Successive levels contain specialist MPUs with increasingly refined models and plans. Of those MPUs responding to an input in the open state, the SELECTOR may choose the one with the highest specificity level. Such an architecture supports learning of new MPUs in the manner described by Holland [4].

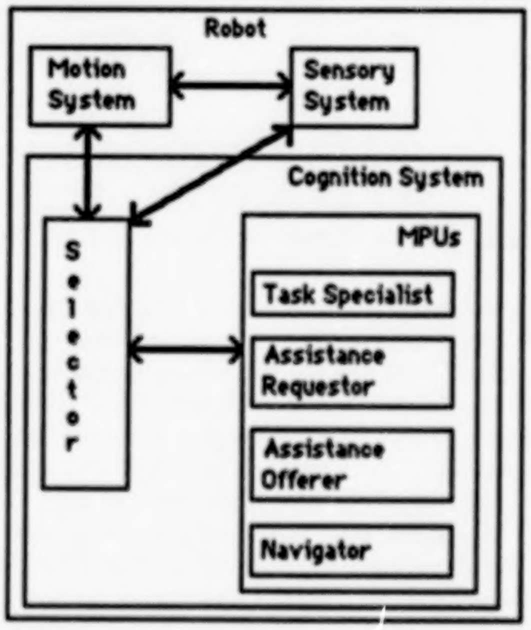


Figure 3. Prototype Robot Configuration

The MPUs comprising the robot brain are of two kinds: those specialized for carrying out specific laboratory tasks and those specialized for more general tasks involving communication, motion, co-operation, etc. A prototype minimal configuration illustrated in Figure 3 contains two robots each with the following MPUs:

Task Specialist MPU: specialized for executing a particular experiment related task, requests help when needed in performing this task by relinquishing control to the Assistance-Requestor.

Assistance-Requestor: MPU specialized for the task of requesting help from other robots. When it is activated, it initiates a protocol which tries to make contact with robots within its range and to engage one which can provide the needed assistance.

Assistance-Offerer: MPU specialized for the task of dealing with incoming requests for help emitted from Assistance-Requestors of other robots. When activated, it decides if help can be offered, and if so, engages in a dialogue with the Assistance-Requestor of the help-seeking robot and sets up a rendezvous. It relinquishes control to the navigator to bring the ROBOT to the requestor's work site. The assistance offer is most easily activated when the ROBOT is idle, and the SELECTOR is in its open state. To replace an already activated MPU (in its closed state), the latter MPU must be able to accept incoming requests for help and relinquish control.

Navigator: MPU specialized for directing the motion sub-system to bring the robot to a given destination. It requests the current motion state from the motion component, and sends it new parameters (direction, speed, and time-step) for travelling to the vicinity of the destination. Once there, it directs the motion component in physically contacting the object or robot at the destination. The touch channel is used for judging when contact has been made.

Elementary scenarios in which the model has been tested are: a) one robot requests assistance, one robot available to offer help; b) one Assistance-Requestor, two Assistance-Offerers available; and c) two Assistance-Requestors, one Assistance-Offerer available. In case b), the first offerer to respond engages with the requestor. The second one receives no confirmation and returns to its previous state. In case c), the first requestor to engage with the offerer is helped. The second one continues to send out assistance requests.

The MPUs are developed as objects in the class "forward-models" of DEVS-Scheme. Models in this class are specified in a rule-based programming paradigm. As shown in Figure 4, a rule, called an activity, is a structure which contains condition and action slots, as usual, and in addition, slots for specifying outputs to be produced before and/or after the action is performed. An action specifies a change in the state of the model. Rules for specifing both internal and external transitions have the same format. Internal transition rule conditions test the phase and state of the model. External transition rules include tests of the input and elapsed time in their conditions.

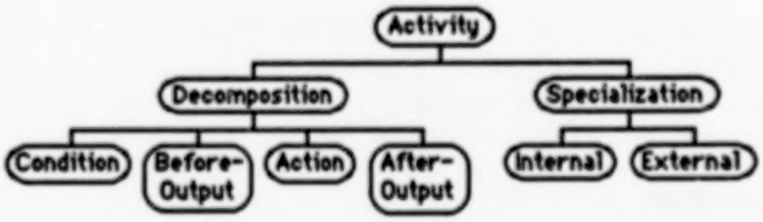


Figure 4. Structure of an activity, i.e., a rule for prescribing MPU state transitions

As an example, an informal presentation of some of the rules for the Assistance-Requestor is given below:

external activity:

```
R1. if phase is wait-for-info
    and receive x on port motion-info
      then record value of x as current position
    and hold-in phase active for 1 unit
```

internal activities:

```
R2. if phase is active
    and need help in executing task
      then send out request for help
    and passivate in wait-for-help
R3. if phase is active
      then send to port starting
    and hold-in phase working for 100 units
    and send to port finished
R4. if phase is working
      then passivate
```

Rule R1 is an external activity which activates the model when an external event arrives on the port "motion-info" while the model is in the phase "wait-for-info". Rules R2, R3, and R4 are internal activities associated with rule R1. Rules R2 and R3 provide alternative courses of action that follow once R1 has placed the model in the "active" phase. R2 starts a sequence of activities dictating what to do if help is needed. R3 bypasses this request for help, and immediately lets the model proceed to the "working" phase. R4 dictates what happens while the model, with or without help, has completed its "working" phase (namely nothing, since the activity itself has not been modelled so far except for the time it takes to execute it).

Rule R3 provides an example where both before- and after-outputs are specified. The before-output is generated just before the action is evaluated while the after-output is generated at the end of the interval specified by the hold-in primitive.

The inference engine underlying forward-models evaluates the rules in the order in which they are added to the model. One advantage of employing rules is apparent in the above example: rules, be they internal or external activities, that are closely associated can be placed contiguously. This avoids breaking sequences of external and internal transitions apart, and thus, aids model comprehension. A second benefit: since outputs may be specified within the rules, the output specification is not separated from the transition specification as necessitated otherwise.

CONCLUSIONS

As a theory of cognition, the above model has the following properties:

a) except at MPU selection points, attention is focused on only those aspects of the environment dictated by the currently activated MPU. If a recording mechanism were to be added which is sensitive only to the current activity, the system, for the most part, would only be able to recall highly restricted portions of its sensory input history (selective attention and recall).

b) while an MPU behavior lies within its envelope, no other MPU can supplant it, even if its initialization conditions better fit the current situation (cognitive hysteresis).

One of the primary goals of the project will be to judge whether these principles provide a workable basis for intelligent robot design. For example, such robots may be so single-minded as to be incapable of flexibly responding to an unknown or changing environment.

Likewise, handling of interruptions, such as requests for help (see below), must be encoded in each MPU since the selector does not inspect inputs in the closed state.

REFERENCES

[1] Bobrow, D.G. (1985). *Qualitative Reasoning About Physical Systems*, MIT Press, Cambridge, MA.

[2] Davis, P.K. (1986). "Applying Artificial Intelligence Techniques to Strategic-Level Gaming and Simulation", In: *Modelling and Simulation Methodology in the Artificial Intelligence Era*, M.S. Elzas, T.I. Ören, B.P. Zeigler (Eds.). North Holland, Amsterdam.

[3] Hardt, S.H. (1988). "Aspects of Qualitative Reasoning and Simulation for Knowledge Intensive Problem Solving", In: *Modelling and Simulation Methodology: Knowledge System Paradigms*, M.S. Elzas, T.I. Ören, B.P. Zeigler (Eds.). North Holland, Amsterdam.

[4] Holland, J.H. (1986). "Escaping Brittleness: The Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems", In: *Machine Learning: An Artificial Intelligence Approach*, Vol. II, R.S. Michalski, J.G. Carbonell, and T.M. Mitchel (Eds.), Morgan-Kaufmann Pub. Co., Los Altos, CA.

[5] Klahr, P. (1986). "Expressibility in ROSS, an Object-Oriented Simulation System", In: *Artificial Intelligence in Simulation*, G.C. Vansteenkiste, E.J.H. Kerckhoffs, B.P. Zeigler (Eds.), SCS Publications, San Diego, CA.

[6] Rajogopalan, R. (1986). "The Role of Qualitative Reasoning in Simulation", In: *Artificial Intelligence in Simulation*, G.C. Vansteenkiste, E.J.H. Kerckhoffs, B.P. Zeigler (Eds.), SCS Publications, San Diego, CA.

[7] Reddy, Y.V., M.S. Fox, and N. Husain (1985). "Automating the Analysis of Simulations in KBS", Proc. SCS Multi-Conference, San Diego, CA.

[8] Reddy, Y.V., M.S. Fox, N. Husain, and M. McRoberts (1986). "The Knowledge-Based Simulation System", *IEEE Software*, March, pp 26-37.

[9] Rozenblit, J.W. and Y.M. Huang (1987). "Constraint-Driven Generation of Model Structures", Proc. Winter Simulation Conf., Atlanta, GA.

[10] Rozenblit, J.W., S. Sevinc and B.P. Zeigler (1986). "Knowledge-Based Design of LANs Using System Entity Structure Concepts", Proc. Winter Simulation Conf., Washington, D.C.

[11] Rozenblit, J.W. and B.P. Zeigler (1988). "Design and Modelling Concepts", In: *Encyclopedia of Robotics*, R. Dorf, S. Nef (Eds.), J. Wiley & Sons, New York.

[12] Sevinc, S. and B.P. Zeigler (1987). Entity Structure Based Design Methodology: A LAN Protocol Example, Tech. Rep. AIS-4, CERL Lab., Dept. of ECE, Univ. of Arizona, Tucson, AZ 85721.

[13] Zeigler, B.P. (1984). *Multifacetted Modelling and Discrete Event Simulation*, Academic Press, London.

[14] Zeigler, B.P. (1986). DEVS-Scheme: A Lisp-Based Environment for Hierarchical, Modular Discrete Event Models, Tech. Rep. AIS-2, CERL Lab., Dept. of ECE, Univ. of Arizona, Tucson, AZ 85721.

[15] Zeigler, B.P. (1987). "Hierarchical, Modular Discrete Event Modelling in an Object Oriented Environment", *Simulation J.*, November.

[16] Zeigler, B.P. (1987). "Knowledge Representation from Newton to Minsky and Beyond", *Applied Artificial Intelligence*, 1, January, pp 87-107.

[17] Zeigler, B.P. (1988). "Implementation of Methodology Based Tools in the DEVS-Scheme Environment", In: *Modelling and Simulation Methodology: Knowledge System Paradigms*, M.S. Elzas, T.I. Ören, B.P. Zeigler (Eds.), North-Holland, Amsterdamm.

[18] Zeigler, B.P. (in press). "The DEVS Formalism: Event-Based Control for Intelligent Systems", to appear in special issue of Proceedings of IEEE.

[19] Zeigler, B.P. and G. Zhang (1987). "Formalization of the System Entity Structure Knowledge Representation Scheme: Proofs of Correctness of Transformations", In: *AI, Simulation, and Modelling*, L. Widman, D. Reidel (Eds.), (in preparation).

N89-15592

Explanation Production by Expert Planners

Susan Bridges
James D. Johannes

University of Alabama in Huntsville
Computer Science Department
Huntsville, AL 35899

Abstract

Although the explanation capability of expert systems is usually listed as one of the distinguishing characteristics of these systems, the explanation facilities of most existing systems are quite primitive. Computer generated explanations are typically produced from canned text or by direct translation of the knowledge structures. Explanations produced in this manner bear little resemblance to those produced by humans for similar tasks.

The focus of our research in explanation is the production of justifications for decisions by expert planning systems. An analysis of justifications written by people for planning tasks has been taken as the starting point for our research. The purpose of this analysis is two-fold. First, analysis of the information content of the justifications will provide a basis for deciding what knowledge must be represented if human-like justifications are to be produced. Second, an analysis of the textual organization of the justifications will be used in the development of a mechanism for selecting and organizing the knowledge to be included in a computer-produced explanation.

This paper describes a preliminary analysis that has been done of justifications written by people for a planning task. It is clear from this analysis that these justifications differ significantly from those that would be produced by an expert system by tracing the firing of production rules. The results from the text analysis have been used to develop an augmented phrase structured grammar (APSG) that describes the organization of the justifications. The grammar was designed to provide a computationally feasible method for determining textual organization that will allow the necessary information to be communicated in a cohesive manner.

PRECEDING PAGE BLANK NOT FILMED

Introduction

Expert system technology has made impressive strides in recent years. Simple rule-based architectures have given way to sophisticated hybrid systems that support a variety of knowledge representation and reasoning mechanisms. One aspect of "expert performance" that has lagged in development is that of explanation. Although most expert system development tools include facilities for building elaborate graphic interfaces, there are many explanation tasks that are not amenable to graphic presentation. The need for textual explanation seems evident when one considers the vast amount of written documentation that human experts are expected to provide to justify the decisions they make.

The expert system literature generally cites three main purposes for explanation facilities [1, 4, 6, 13]. First, explanation can be used by the knowledge engineer to test and debug the system. Second, explanation assures sophisticated users that the system's knowledge and reasoning process are sound and allows the detection of situations in which the system is being asked to perform a task outside the boundaries of its capability. Finally, explanation facilities can be employed to instruct naive users about the knowledge of the system. These functions relate to the interaction of the user and expert system in the course of a consultation. Another aspect of explanation that will become more important as expert systems are used in complex and critical domains is that of recording justifications of decisions. Human experts are usually called on to provide a written justification for the validity of their decisions. It would appear reasonable, therefore, that a computer program that aids in making these decisions should also provide assistance in providing justifications for the decisions.

Expert system explanations have typically been produced by the use of canned text (for tasks such as defining terms) or by tracing the rules that have fired during the inference process. Systems that generate rule traces usually have some provision for translating the syntax of the rules into a natural language form for presentation to the user. Although this approach offers more explanation capability than is found in traditional computer programs, the traces produced are very different from the explanations one would expect from a human expert. In particular, rule traces are not adequate for answering questions of the form "Why is this a valid decision?" The research described in this paper is based on the observation that when human experts are asked to justify decisions they have made, they do not merely recite the steps taken in reaching the decision.

The limitations of the "trace the rules" approach have been widely discussed in the literature [3, 12, 14]. These limitations are of three types. In the first kind, the traces tend to be very long and contain much information that is of no interest to the user. This problem becomes more acute as the size of the system grows. The second problem with rule traces is the absence of much information that would be expected in a human-produced explanation. For example, information about the problem solving strategy of the system is not present in the rule trace because it is not explicitly represented in the rules and knowledge structures of the system. Other information, such as that needed to define terms, is often absent from the knowledge base altogether. The third type of problem concerns the structure of the explanations produced. Research in natural language processing has shown that multisentential text produced by humans exhibits a characteristic structure and organization that facilitates the communication process [7, 8]. This structure is not found in rule traces.

Much of the research in explanation production done to date has been in systems that perform medical diagnosis or fault diagnosis in electronic or mechanical devices. This research has addressed the first two problems listed above by developing methods for incorporating the knowledge needed for explanation in the knowledge structures of the system and for tailoring the system's responses to the user [3, 13]. The problem of structuring explanatory text has been largely unaddressed.

The goal of our current research is the development of a methodology for selecting and organizing the knowledge that is to be used to construct multisentential explanatory text that is a justification of the recommendations made by an expert planning system. The task of natural language generation has traditionally been divided into two components. A strategic component determines the content and structure of the text while a tactical component determines the natural language surface structure (which words and syntactic structures to use). The emphasis of our research is the strategic component.

Expert planning programs offer an attractive test bed for the production of justifications of decisions. There has been a great deal of research into the development of general frameworks for planning systems [2, 9], but this work has largely ignored questions of explanation.

Analysis of Text

Students are taught from early elementary years that there are ways to organize writing that will increase its effectiveness. It stands to reason, then, that text written for a specific purpose in a limited domain will exhibit more regularity of organization than text in general. Language understanding systems have long made use of this property [15]. For this reason, it was thought that the analysis of text written for a specific type of task could be used as the basis for the development of a grammar formalism that could be used in the generation of similar text by an expert system.

Written, rather than spoken text, was used in the analysis phase of this research for several reasons. First, written text is generally better planned and organized than spoken text. Spoken text often contains partial sentences and ungrammatical constructions that would be unacceptable in written form. In addition, speakers use facial expressions and tone of voice to convey much of their meaning. Spoken text is often directed toward a more specific audience and so requires a more elaborate user model than written text.

The type of task chosen is that of justifying the validity of a plan constructed by an expert planning system. An initial text analysis was done using justifications of Master's degree plans of study written by University of Alabama in Huntsville graduate students. Additional analysis has since been done of justifications of travel itineraries. The justifications of plans seem to follow the general form of identifying the each component of the plan, and the planning constraints that each component satisfies.

A Grammar Describing the Structure of Justifying Text

The augmented phrase structured grammar (ASPG) formalism was chosen as a representation for the text structure. Other approaches to representing text structure that have been used in previous research include fixed semantic patterns [3, 10, 11] and context-free grammars [8, 14]. The use of semantic patterns limits the flexibility of the representation by restricting the number of text structures that can be generated to a small finite set. Although the use of a context-free grammar allows an infinite number of structures to be generated, a mechanism outside the grammar itself must be used to control the application of rewriting rules when more than one applies. The APSG facilities for attaching attributes to non-terminals and conditions to rewriting rules allows generation of an infinite number of structures and provides a method for embedding the control of the application of rewriting rules in the grammar.

After study of the justifications submitted, an attempt was made to develop an APSG that could be used to guide the generation of similar justifications by an expert system. The starting point for text generation is necessarily some representation of the relevant knowledge. It is understood that an explanation system cannot communicate information that the knowledge base does not contain or cannot derive. Thus, one important factor in the development of these systems is the representation of appropriate knowledge. It is assumed in this paper, that the appropriate knowledge is represented, but the method of representation is left undefined.

The APSG formalism uses auxiliary evaluation functions in the assignment of values to attributes and testing of conditions on rewriting rules[15]. In order to use the knowledge of the system to direct the generation of explanations, it was necessary to provide an interface between the grammar and the knowledge base. This is done by use of special auxiliary evaluation functions that access the knowledge base and return values that can be used to direct the application of grammar rules and as building blocks in the message constructed by the generation process. In addition to inherited and synthesized attributes, attributes that derive their values solely from functions that access the knowledge base are called assigned attributes. The Start symbol of the grammar will have one or more attributes that are given values before the generation process begins.

It is assumed that the generation process will proceed in a left to right, depth-first manner and so the restriction is imposed that synthesized attributes can only be inherited from left to right. In addition, the generation process will initially proceed in a top down manner and so attributes cannot be synthesized in a sub-tree and then used in a condition at the root of the sub-tree to test its validity. This means that attributes used in conditions will, in general, be inherited or assigned. When more than two rules can be applied, it is assumed that the conditions on the rules are sufficient to decide which is applicable so that backtracking can be avoided. Evaluation rules for attributes that will be tested in conditions are given before the rewriting rules in which the conditions are tested.

The grammar given is unique in that it does not have any terminal symbols. It is assumed that "non-terminal" symbols, once generated, are never retracted. Thus, when a non-terminal is encountered for which there is no applicable rule, that symbol, in effect, becomes a terminal symbol. The tree built as the rules are applied acts as a framework for the message to be built. Pieces of the message are built at each of the leaves and are brought together as as synthesized attributes from the leaves to the root. The message this built has a structure imposed by embedded lists and can contain additional information such as focus of attention, tense, etc.

No attempt has been made to construct a grammar that can generate all of the organizations found in the justifications that were studied. Rather, the grammar is an attempt to provide a method for providing an organization that is computationally feasible and yet flexible enough to allow the necessary information to be communicated in a planned cohesive manner.

Summary

Human experts are often expected to provide written justification of the decisions they make. As the use of expert systems becomes more widespread, it will be become increasingly important for these systems to have the capability to compose text that justifies their decisions. Traditional "trace of the rules" explanations are not sufficient for this task because they include much information that not pertinent, they omit other information that is typically found in human-provided explanations, and they lack any organizing structure. This paper explores the possibility of using an augmented phrase structured grammar to describe the structure of justifications of expert planning decisions. The grammar provides a mechanism selecting and organizing the information to be provided in the justification.

References

1. Buchanan, Bruce G., and Shortliffe, Edward H. *Rule-Based Expert Systems*. Addison-Wesley, Reading, MA, 1984.

2. Chapman, David. Planning for Conjunctive Goals, *Artificial Intelligence*, 32, 1987, 333-377.

3. Clancey, William J. The Epistemology of a Rule-Based Expert System--a Framework for Explanation. *Artificial Intelligence*, 20, 1983, 215-251.

4. Finin, Timothy W., Joshi, Aravind K., and Webber, Bonnie Lynn. Natural Language Interactions with Artificial Experts. *Proceedings of the IEEE*, 74, 1986, 921-938.

5. Hasling, D. W., Clancey, W. J., and Rennels, G. Strategic Explanations for a Diagnostic Consultation System. *International Journal of Man-Machine Studies*, 20, 1984, 3-19.

6. Hayes-Roth, Frederick, Donald A. Waterman, and Douglas B. Lenat, (Eds), *Building Expert Systems*. Addison-Wesley Publishing Company, Reading, MA, 1983.

7. Mann, William C., and Thompson, Sandra A. Rhetorical Structure Theory: Description and Construction of Text Structures. In: Gerard Kempen (Ed.), *Natural Language Generation*, Martinus Nijhoff Publishers, Dordrecht, 1987, 85-96.

8. McKeown, Kathleen R. *Text Generation*, Cambridge University Press, Cambridge, UK, 1985.

9. Sacerdoti, Earl D. *A Structure for Plans and Behavior*, SRI Technical Report TN-109 SRI International, Menlo Park, CA, 1975.

10. Schank, Robert C., *Explanation patterns: Understanding Mechanically and Creatively*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

11. Schulman, Robert, and Hayes-Roth, Barbara, Plan-Based Construction of Strategic Explanations. Knowledge Systems Laboratory, Report No. KSL 88-23, Stanford University, Stanford, CA, 1988.

12. Swartout, William R., XPLAIN: a System for Creating and Explaining Expert Consulting Systems, *Artificial Intelligence*, 21, 1983, 285-325.

13. Swartout, William R. Knowledge Needed for Expert System Explanation, *Future Computing Systems*, 1, 1986, 99-114.

14. Weiner, J. L. BLAH, A System Which Explains its Reasoning. *Artificial Intelligence*, 15, 1980, 19-48.

15. Winograd, Terry, (1983) *Language as a Cognitive Process: Syntax*, Addison-Wesley, Reading, MA.

N89-15593

KNOWLEDGE REPRESENTATION ISSUES FOR EXPLAINING PLANS

Mary Ellen Prince
James D. Johannes

University of Alabama in Huntsville
Computer Science Department
Huntsville, AL 35899

ABSTRACT

Explanations are recognized as an important facet of intelligent behavior. Unfortunately, expert systems are currently limited in their ability to provide useful, intelligent justifications of their results. We are currently investigating the issues involved in providing explanation facilities for expert planning systems. This investigation addresses three issues: knowledge content, knowledge representation, and explanation structure.

Introduction

An important characteristic of an intelligent system, whether human or computer, is the ability to explain or justify its actions. Recognizing this fact, expert system developers were the first to regularly incorporate explanation facilities into their programs. [3]. Unfortunately, early attempts at automated explanation produced results that were significantly different from human explanations, both in organization and in information content.

In large measure this was due to the methods which were used to model expert problem solving knowledge. Production rules, the most popular form of knowledge representation, proved to be an effective representation for generating solutions, but were less than satisfactory for generating explanations of those solutions. The domain principles and expertise which determined the organization and content of the rule base were represented implicitly, if at all, and therefore could not be used to justify the system's behavior.

Recent research has focussed on methods of improving the explanation capability of intelligent systems. ([4], [6], [9], [10], [11]). At least three issues must be considered: first, identification of the kinds of knowledge which constitute a useful explanation; second, determination of a representation formalism to make the knowledge readily accessible to the explanation generator; and third, methods for selecting and organizing the knowledge in order to present it in a meaningful format.

This paper will address the first issue in depth and will suggest an approach to the second. For a discussion of explanation organization, see [2]. The explanation domain will be planning systems.

The Importance of Explanations

There is a significant gap between what users would like to see in an explanation system and what is feasible in light of current theory and technology. Although researchers are working to close the gap, much remains to be done.

A system that is able to explain its own behavior has several advantages over systems that lack this ability. For example, adequate explanation facilities can reassure skeptics that the system's reasoning processes are sound and its results are reliable. They can also serve a tutorial purpose. A properly constructed description of the strategy and domain principles used to derive a solution can provide insight which the user can then apply to other, similar, problems. The novice is thus encouraged to expand his knowledge of the domain, much as if he were working directly with a human expert. Explanations can be useful to knowledge engineers during the test and debug phase of system development, just as program traces are useful to a programmer under similar conditions ([1], [6]). In addition, good explanations can provide an automatic documenting capability. Finally, it can be argued that a system which contains the knowledge needed to produce good explanations can also be designed to use this information to improve its own performance in areas such as error recovery.

Current State of Explanation Technology

Explanation technology is severely limited in its ability to provide the benefits cited above. Explanations typically assume one of two forms: natural language traces of the rules currently under consideration or, less frequently, "canned text" inserted by the designer.

Rule traces have some advantages. They provide an accurate record of the program's activity and are thus helpful for debugging the knowledge base and for showing when the program is being pushed beyond the limits of its ability. In addition, modifications to the rules are automatically reflected in the associated explanations, thereby insuring consistency. Explanations produced by paraphrasing rules suffer, however, from several defects. They are poorly structured, and often filled with low level operational details that are of little interest to the average user. More seriously, they are in general incapable of explaining causal relations, the rationale behind a solution, strategic issues, or anything about fundamental domain principles. The reason for this is that the knowledge required for deep explanations is not explicitly represented in the rules. Problem solving strategy and domain relations are implicit in the clause ordering of rule concepts, and the principles which justify the rules and strategy are missing altogether.

Canned text explanations can be used to annotate individual rules or groups of rules, but this approach lacks the flexibility which is needed for a full scale explanation system. It is difficult to anticipate every explanation which may be required. In addition, there is no guarantee that changes to the program code will be reflected in changes to the associated explanations, since there is no automatic connection between the two.

In summary, it is apparent that the current state of explanation technology falls far short of that which may be desired. The following sections will examine the kinds of questions that a good explanation system might reasonably be expected to address, and identify the knowledge needed to respond to these questions. For background, a discussion of the planning domain will be presented.

The Planning Domain

This paper addresses explanation generation in the context of expert planning systems. Previous explanation research has concentrated on diagnostic expert systems, primarily in the areas of medicine and electronic trouble shooting. An important consideration is whether these findings can be extended to other domains, such as planning. A comparison between planning systems and diagnostic systems will help to answer this question.

Simply stated, an expert planner is a system that generates a sequence of steps which, when applied from a given starting state, will produce the desired goal state. Some systems are interactive, so that feedback during plan execution can influence future planning decisions. In traditional planners, often called strategic planners, it is more commonly the case that plan generation and plan execution are two distinct processes. Robot problem solvers and automatic programming are two common application domains for planning systems.

The individual steps in a plan are produced by plan operators, which describe the legal actions or events that can occur in the application domain. Operators are typically described by a set of preconditions, which determine when an operator can legally be applied, and a set of postconditions, which describe the operator's effect on the world state. Operator descriptions may also include additional information such as procedures for accomplishing desired effects, ordering constraints, and resource requirements.

The planning process consists of choosing appropriate operators and ordering them to achieve the goals which constitute the final state. Early planners were linear; that is, they developed a sequence of steps to achieve each individual goal in order. At every point in the planning process the plan was fully detailed, but complete only to that point. STRIPS [7] is the most familiar example of a linear planner.

Hierarchical planners, on the other hand, start with a high level representation of the entire plan and refine it through several levels of abstraction to arrive at the final sequence of primitive operators. A feature frequently associated with hierarchical planners is partial ordering of actions. Non-hierarchical planners are often forced to make arbitrary ordering decisions in order to maintain the linear nature of the incomplete plan. Hierarchical planners postpone committment until it is clear that the committment will not have to be undone at some later stage. NOAH[8], NONLIN[12], and SIPE[13] are hierarchical planners.

A third paradigm is case-based or script-based planning. This approach relies on a library of existing skeletal plans which are adapted to new situations by various refinement and "debugging" techniques. HACKER and MOLGEN[5] are two examples of planners that fit this paradigm.

Comparison of Planning Systems and Diagnostic Systems

Comparison of planning systems and diagnostic systems suggests several parallels that can be exploited to transfer explanation theory from the diagnostic domain to the planning domain. Two that will be investigated here are the knowledge bases and the inferencing processes.

The knowledge of a typical diagnostic system is encoded in production rules, a simple, flexible formalism for representing expert reasoning. Rules consist of two parts: the premise, typically a conjunction of clauses, and the conclusion, or goal. The conclusion can be established by proving that the clauses in the premise are true. This may be done by gathering evidence directly or by proving other rules which have the same clauses as goals. Thus the knowledge base can be viewed as a hierarchical network with implicit links between goals and premises.

The operators in a planning system serve a function analagous to that of production rules in the sense that they contain the necessary problem solving knowledge. The analogy can be extended to structural aspects of the knowledge as well. An operator's preconditions (premises) must be satisfied in order to achieve the desired postconditions (goal). Satisfying preconditions can be accomplished by applying other operators with the appropriate postconditions, thus giving the set of defined operators a network structure.

Inference in expert systems is accomplished by rule chaining. Depending on its design, a system may chain backward from a suspected diagnosis to known evidence, or it may chain forward from the evidence to a diagnosis.

Planners, particularly those based on the hierarchical paradigm, may use problem reduction as an inference technique. The method is to first express a plan as a sequence of high level goals and then to refine each abstract goal into a set of more concrete subgoals. The refinement process can be repeated as often as is necessary to produce the final sequence of primitive actions. As an alternate, a planner might employ means-ends analysis. This approach involves a comparison between the current state and the goal state. Wherever differences are detected, operators are selected to reduce the differences. This is also an iterative process.

Plan derivation and diagnosis differ in the details of how the appropriate operators or rules are selected but the effect of the selection process is similar in both cases. At any point during the inferencing process there exists a stack of goals, implicit or explicit, which must be realized. An achieved solution represents a path through the network of rules or operators. For planning systems the path is equivalent to the plan; for diagnostic systems it represents the chain of reasoning that led to a specific diagnosis.

The solution produced by a planning system is more complex than the solution produced by a diagnostic system. A plan is a structured entity consisting of an ordered sequence of steps, while a diagnosis consists of a single entity. In addition, many planners operate in dynamic, multi-agent domains. They must plan simultaeous actions, prevent harmful interactions between competing agents, and consider the effects of actions over which they have no direct control. It is reasonable to expect that this added complexity will cause a corresponding increase in the complexity of the associated explanation. The next section will expand on this premise through a discussion of the epistemological issues of explanation theory as applied to expert planners. It will first outline some of the issues that must be addressed by an explanation system and will then present a taxonomy of explanation related knowledge.

The Epistemology of Explanations

A good explanation facility should be flexible enough to meet the needs of domain experts, novice users, and system designers. The following items illustrate the kinds of questions that it might be expected to address.

- Domain Facts, Principles, and Terminology
  Terminology is important as a foundation for understanding higher level explanations. Principles describe the problem solving procedures which can be used to achieve a goal. In well defined domains most facts can be expressed as causal relations, while in less formalized domains, empirical associations and heuristics play an important role.
- Comparisons and Choices
  Explaining why one action is preferable to another is a difficult task. Such decisions may be predicated on an accumulation of prior evidence, or on anticipation of future effects. In general, this kind of explanation requires a knowledge of constraints, interactions between events, and ultimate goals.

- Justification
  Justifying a single step in a plan can be as simple at stating a causal relationship or as complicated as explaining a choice. Justifying an entire plan may require the system to identify strategies, constraints, priorities, resource restrictions, and temporal issues.
- Methodology
  Questions about methodology refer to the mechanics by which a particular solution was obtained.
- General Strategy
  In addition to specific methods, most problem solvers also rely on abstract principles and weak methods to guide the problem solving process.

No system has yet been able to respond to all of these issues. Expert systems have traditionally answered questions about methodology by paraphrasing a chain of executed rules. In planning systems, a similar effect can be achieved by showing how operators in a general procedure have been instantiated with case-specific data. MYCIN[3] had a limited ability to compare alternative drug therapies. In NEOMYCIN Clancey[4] and Hasling et al [6] extended the explanatory capabilities of MYCIN by incorporating meta-rules to provide information about strategy. Shulman and Hayes-Roth[9] designed an explanation module to provide justifications and feasibility evaluations for certain knowledge systems where the reasoning was controlled by a strategic plan. In general, however, most systems lack the deep knowledge required to provide a broad range of explanations.

The remainder of this section identifies the kinds of knowledge needed for plan explanation. This identification is based on previous explanation research from the diagnostic domain, as well as on the specific needs of the planning domain. For purposes of discussion, the knowledge will be classified as either meta-knowledge (knowledge about knowledge), domain knowledge, or case-specific knowledge.

- Meta-Knowledge

Meta-knowledge embodies knowledge about control strategy and problem solving techniques. While a specific method or strategy might be classified as domain knowledge, the guidelines used to choose between competing strategies fall into the category of meta-knowledge. Many of the so-called "weak methods" can also be categorized this way. Examples of meta-knowledge are "Look for common causes for a device malfunction before looking for unusual causes" or "Avoid ordering plan operators until there is a reason to do so."

It is not clear that there are principles which are applicable to every planning domain. For example, a linear planner might employ the principle "Order plan operators arbitrarily if no information exists; modify later if necessary" instead of the "avoid ordering" principle cited earlier. It is clear, however, that every planning system operates on a set of general strategic principles which may, in fact, have wide application.

Some illustrations of these general strategies may be found in the literature. Swartout[10], for example, recommends the use of "tradeoffs" and "preferences", where tradeoffs indicate the pros and cons of selecting a particular goal-achieving strategy and preferences are used to prioritize goals. Planners in multi-agent domains that permit parallel actions have devised methods for resolving the conflicts that arise when actions in one branch of a plan interfere with actions in another branch[13]. System designers must identify the abstract principles that guide their own problem solving and incorporate them into the meta-level knowledge structure. In order to provide good explanations of general strategy and to justify final plans it is important that the information be represented explicitly.

- Domain Knowledge

Without domain knowledge, it is impossible to explain terminology, principles, and domain facts. It is also difficult to furnish justifications and explanations of general principles unless domain specific information is available. Both declarative and procedural knowledge are required here. Declarative knowledge encompasses terminology and factual information, while procedural knowledge expresses how goals can be accomplished. Swartout and Smoliar [11] discuss the need for terminological, domain descriptive, and problem solving knowledge in the context of EES, an expert system which diagnoses cardiac difficulties and prescribes digitalis therapy. Their structure is sufficiently general to apply to planning as well as diagnostic domains.

The terminology of a planning system includes all domain concepts. Physical objects, their properties, and relations among objects such as "on-top-of" or "greater-than" must be defined in terms of system primitives. Factual knowledge can be represented as assertions of causal relations or probabilistic associations. Certain types of constraints which control the temporal ordering of operators and specify harmful or helpful interactions may also be represented this way.

Procedural or strategic knowledge in intelligent planners involves the selection and ordering of plan operators. The application of domain strategies is subject to control by meta-level knowledge and is, at the same time, dependent on case-specific information that can activate constraints and ordering rules. To be fully explainable, strategies must also be supported by a rationale based on domain facts.

- Case-Specific Knowledge

Every instance of a planner's operation begins with a specification of the initial world state, the desired goal state, and a list of constraints, availabe resources, and other pertinent information. Using meta-level and domain strategies, the planner then generates a sequence of steps which describe how to achieve the goal. The final plan consists of these steps, instantiated to satisfy the initial specifications.

While the plan itself may be used to explain methodology, much as a traditional diagnostic system uses its rule chain to explain its diagnosis, it is necessary to keep a case history of the problem-solving process in order to provide deep explanations. At a minimum, the case history must include the procedures used, choices made, and the reasons for those choices.

As has been previously noted, some choices occur when the planner is forced to decide among two or more operators. Other decisions determine the ordering of plan steps. Diagnostic systems use certainty factors or other numerical weights as an aid when making similar decisions. Quantitative values do not contain enough information to generate satisfactory explanations, however, nor are they always appropriate in the planning domain. Planning decisions result from a combination of constraints, goal priorities, resource availability, or the knowledge that one or more of the options would interfere with the achievement of some future goal. This is the type of knowledge that must be kept in the case history.

It should be clear from the preceding discussion that there is no absolute boundary separating meta-knowledge, domain knowledge, and case-specific knowledge. Furthermore, there are situations where it is necessary to integrate information from more than one knowledge level to produce an adequate explanation. The next section will investigate methods of structuring planning knowledge to make it accessible to the explanation generator.

Representation Issues

The knowledge required to explain plans is, on the whole, the same knowledge that is required to generate the plans. Previous intelligent systems have made much of this knowledge unavailable for explanation generation. The problem now is to develop representation formalisms that will make the information explicit without unduly affecting the efficiency of the plan generator. A completely developed representation scheme is beyond the scope of this paper. Instead, it will concentrate on outlining a general knowledge structure to guide future research.

Domain terminology is best represented as a type hierarchy of nodes. The highest levels of the hierarchy serve as an index to domain concepts, while the lowest level can be instantiated with case-specific data. Individual nodes have attributes which can be either pointers to other nodes, definitions, or other properties. The pointers define the hierarchy and permit property inheritance. Attributes describe concept features and may be used to record constraints on the values of plan variables.

Operators also have a natural hierarchical structure. Abstract operators encode meta-level strategies which in turn invoke domain procedures. At the bottom of the hierarchy are the primitive operators which define individual plan steps. In addition to parameters, pre-conditions, and post-conditions, operators should include information about constraints, resources, and rationales. Constraints may apply to variable values or to temporal ordering. Resource requirements describe the domain resources needed to perform the step and the duration for which the resources must be available. The rationale may state that the operator is necessary in order to establish some condition needed for a future action or it may provide a causal justification for the process invoked.

The case history records the refinement process by which the plan was generated, giving it, too, a hierarchical structure. The highest levels contain information about meta-level decisions, such as options between alternative strategies. Intermediate levels are concerned with domain dependent choices. The lowest level corresponds to the actual steps of the plan. Nodes in the history are instantiated with case-specific data, where appropriate. Choice nodes can be annotated with reasons that justify the choices. For explanation purposes it is vital that the domain facts and definitions that motivated the choices be represented. The structure will then contain all knowledge needed to justify the plan.

6. Conclusion

Explanation theory is just beginning to move beyond the narrow scope of early efforts. Providing intelligent responses to a variety of questions requires a full and explicit representation of the knowledge involved. The hierarchical nature of knowledge in an expert planning system enables the planning process to be explained on many levels of abstraction. Systems which rely primarily on the knowledge embedded in low-level rules forfeit this opportunity. Building the knowledge bases required for adequate explanations is no small task. It is an activity that requires careful attention from domain experts and knowledge engineers alike. The result of this effort, however, is a system that will be more responsive to the needs of its users.

REFERENCES

1. Berry, D. C. and Broadbent, D. E. "Expert Systems and the Man-Machine Interface", Expert Systems, vol. 4, no. 1, Feb. 1987, 18-28.

2. Bridges, S. and Johannes, J. D. "Explanation Production by Expert Planners", Proceedings of Fourth Conference on Artificial Intelligence for Space Applications, to be published. 1988.

3. Buchanan, B. G. and Shortliffe, E. H. Rule Based Expert Systems. Addison-Wesley, Reading, MA, 1984.

4. Clancey, W. J. "The Epistemology of a Rule-Based Expert System - A Framework for Explanation", Artificial Intelligence, 20, 1983, 215-251.

5. Cohen, P. and Feigenbaum, E. A. The Handbook of Artificial Intelligence, vol. 3. William Kaufman, Inc. Los Altos, CA, 1984.

6. Hasling, D. W., Clancey, W. J. and Rennels, G. "Strategic Explanations for a Diagnostic Consultation System". International Journal of Man-Machine Studies, 20, 1984, 3-19.

7. Nilsson, N. J. Principles of Artificial Intelligence. Tioga Publishing Co., Palo Alto, CA, 1980.

8. Sacerdoti, E. D. A Structure for Plans and Behavior. Elsevier North-Holland, Inc. New York, NY, 1977.

9. Schulman, R. and Hayes-Roth, B. ExAct: A Model for Explaining Actions. Knowledge Systems Laboratory, Report No. KSL 87-8, Stanford University, Stanford, CA, 1987.

10. Swartout, W. R. "Knowledge Needed for Expert System Explanation", Future Computing Systems, vol. 1, no. 2, 1986, 99-113.

11. Swartout, W. R. and Smoliar, S. W. "On Making Expert Systems More Like Experts", Expert Systems, vol. 4, no. 3, Aug. 1987, 196-207.

12. Tate, A. "Generating Project Networks", Proceedings IJCAI 77, 1987, 888-893.

13. Wilkins, D. E. "Domain-Independent Planning: Representation and Plan Generation", Artificial Intelligence, 22, 1984, 269-301.

N89-15594

Simple Explanations and Reasoning: From Philosophy of Science to Expert Systems*

Daniel Rochowiak
205-895-6217
Johnson Research Center — Philosophy
University of Alabama in Huntsville
Huntsville, Alabama 35899

ABSTRACT

Explanation facilities are an important extension of the rule based paradigm. By using contrast why questions and a more textured notion of reasoning, a robust schema for simple explanations can be developed.

INTRODUCTION

At first glance it seems rather easy to characterize explanation. An explanation is a deductive argument that satisfies the conditions of empirical adequacy. [7] However, behind this apparently simple account can be found a great many issues. One of these concerns the pragmatics of explanation; deductive explanations often fail to explain anything to the person seeking the explanation. I will examine the pragmatic dimension of explanation and indicate how a more 'textured' notion of reasoning can enhance the explanation facilities of the expert system paradigm. The domain of interest will be a general one in which there are objects, states of objects and causal paths between the objects.

Among the facilities commonly found in expert system building tools are the why? and how? explanation facilities. Typically the why? facility is engaged at a prompt and reports the rule that is currently being examined, while the how? facility requests the user to identify a particular parameter for which the system has set a value and reports the rule by which it was set. Such facilities operate in the style of deductive explanation. In both there are conditional claims (laws or rules) together with specified conditions (initial conditions or user entered values) that deductively lead to particular conclusions (the explanandum, or the values of parameters). Deductive explanations have been criticized for attending more to the grounds of an explanation than the particular explanation of a given event. [8] Similarly the how? and why? facilities attend more to the rules of the system than to the events to be explained. The sorts of explanation offered within the expert system should not be confused with the sorts of explanations that might be considered ordinary in other contexts. The explanation facilities might explain, for example, why one would come to think that a particular part failed, but would not explain why the part failed. Such criticism points to the importance of pragmatic considerations in explanation.

VARIETIES OF EXPLANATION

Explanations come in many forms. Scientific explanations are one well studied group of explanations. One form of scientific explanation proceeds from a scientific law, theory or model to a deductive account of a phenomenon. Although Hempel's original formulation of such deductive nomological (DN) explanations has been much criticized, it provides both a good starting point and a base that, with suitable extensions and amendments, can capture a large range of scientific explanations.

The DN model of scientific explanation invites comparison to the notion of backward chaining in expert systems. The explanandun is known and the collection of scientific principles is searched in an effort to find the conditions which, if satisfied, would constitute the explanans. The collection of principles retrieved along with the specified conditions constitute the explanation of the phenomenon as described in the explanandum. Thus, the pattern of such an explanation would be:

<explanandum asserted as true> because <explanans retrieved by backward chaining>.

Within the range of DN explanations, a distinction must be drawn between the epistemic and ontic modes of explanation. The epistemic mode employs the sort of reasoning, captured in rules, that an expert or scientist would use in solving a problem or producing an explanation. The ontic mode concentrates on the scientific principles and laws which, if true, would produce a sound deductive argument with the explanandum as the conclusion. Taking a liberal approach to both scientific explanation and expert systems, it will be assumed that the epistemic mode can be considered to be the locus of explanatory activities. Thus, the explanation produced by the operation of an epistemic system will provide the reasons for asserting that the explanandum is true.

Accepting the epistemic mode as primary suggests that the operation of an expert system itself can be taken as an instance of explanation. However, more is required since what is often at issue is either why certain rules are used or why there are such rules at all. A simple approach to this can be taken by adding the idea that rules themselves are often linked to some backing that explains the rule. Though this is only a small deviation from the epistemic mode of DN explanation, it would provide for richer, more informative, explanations.

Other explanatory patterns require a greater divergence from the DN model.

In some cases the focal point of explanation is why one member of a particular kind behaved in a way that the other members of that kind did not. For example, one might want to explain why two parts of the same kind came to be in different states. Here it seems reasonable to think that an explanation is provided by a list of the property differences between the two instances. The pattern of such an explanation is unlike the DN pattern since the focus is difference and not deduction. The pattern of such an explanation would be:

<differences in instances> because <differences in kinds>.

In other cases the focal point of the explanation is a temporal or causal account of a how an object came to be in a particular state. Such explanations are akin to historical explanations in which one is searching for a significant event that leads to a particular outcome. Such explanations are unlike DN explanations since the focus is temporal and not logical. The pattern of such an explanation would be:

<state of object> because <chain of events>.

The variety of explanations in the context of scientific reasoning strongly suggests that there will be more than one explanation pattern. In turn this suggests that in the pragmatics of explanation attention must be paid to determining what sort of explanation is desired.

WHY QUESTIONS AND THE VARIETIES OF EXPLANATION

One can request an explanation in many different ways. One might ask 'Why does parameter P have value V?' or one might ask 'How is it that the system is now asking this question?' The former although asked as a why question is the province of the how? facility and the latter, though asked as a how question is the province of the why?

facility. For the sake of clarity and convenience I will assume that all requests for explanation can be represented as why questions, and suggest that all appropriate requests for explanation embody contrast why questions, 'Why is it this rather than that?'

In the simple question 'Why P?' P represents the surface topic (ST) of the question and its assumed that P is true. However, if a context is not invoked, such questions are ambiguous. [10, 1] The question 'Why did part-234 fail?' is ambiguous. At the level of the expert system (ES), it might be a request for an explanation of why the system inferred that part-234 failed rather than asking for a test to be performed on part-234. At the level of the causal process (CP), it might be a request for an explanation of why part-234 rather than part-123 failed, or a request for an explanation of why part-234 failed rather than continued to operate. Thus, why questions should be understood as making reference to some contrast class. 'Why P?' is a specialization of 'Why P rather than Q?' when the contrast class is already understood. In the more general form 'P rather than Q' is the intended topic (IT) of the explanation, and it is assumed that P is true and Q is false.

The contrast class for the IT must contain at least P and Q, but may contain other propositions. The propositions contained in the contrast class may be exclusive, inclusive or unspecified. If they are exclusive, then showing why P is true, will amount to explaining why Q is false. If they are inclusive, a separate account will be needed to show why Q is false. If they are unspecified, then the question should be treated as if it were an ST question.

The particular items that appear in the contrast space can be generated by examining the instances or causal paths. If the IT refers to things that have instances, then the space would include all the instances of that type. For example, if the IT referred to the contrast of part-234 and part-123 and both parts were of the type philosophator, then the contrast space would be composed of all instances of philosophators. Alternatively, if the IT referred to the state of an object, then the space would be composed of all of the paths through the part. For example, if the IT referred to the failure of part-234 rather than its continued operation, then the contrast space would be composed of all the paths through part-234. It should be noted, however, that the state of a device could be determined by inference or by direct measure. If the state is determined by direct measure and cannot be inferred by the rules of the the ES, then the cause of the part's state will be labeled as internal. If the state is either inferred by the rules or is measured but can be inferred from the rules, then it will be labeled as external.

The integration of contrasting why questions with the varieties of explanatory patterns provides an environment in which the user can receive more meaningful explanations. Although the explanations are a bit simple they do respond to the context and provide for multiple explanatory views.

Diagram 1 summarizes the discussion to this point. Simply put the system is designed to have the user refine the ST to an IT. The system then checks for the truth of the assumptions, and if they are false issues a corrective explanation. The user then determines the level of the explanation, ES or CP. If the level is ES, then an ES explanation is assembled. If the level is CP, then the contrast spaces are constructed either by finding the instances of a particular kind of object or by finding all the paths through the object. Appropriate CP explanations are assembled unless the type of contrast is unspecified. If the type of contrast is unspecified, then the system returns for a further clarification of the IT.

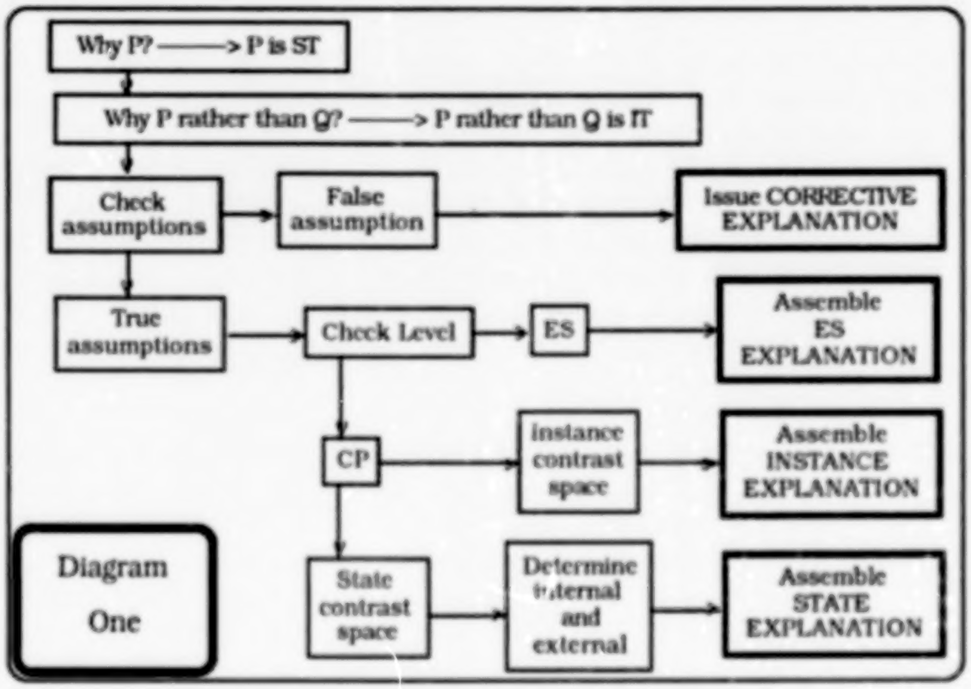
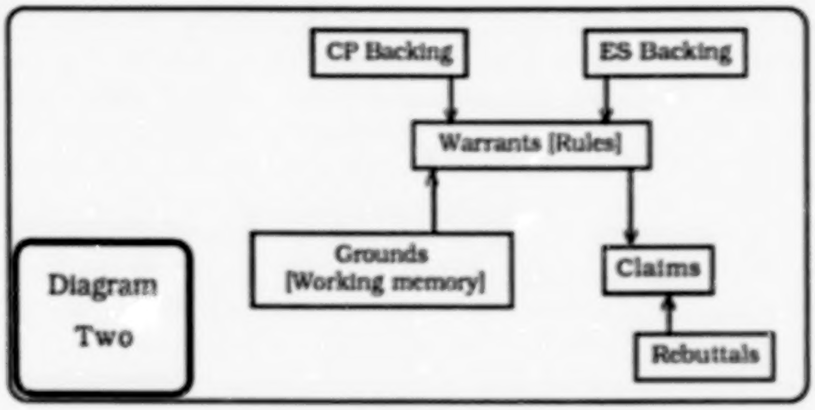

Why P? ———> P is ST
Why P rather than Q? ———> P rather than Q is IT
Check assumptions
False assumption
Issue CORRECTIVE EXPLANATION
True assumptions
Check Level
ES
Assemble ES EXPLANATION
CP
instance contrast space
Assemble INSTANCE EXPLANATION
State contrast space
Determine internal and external
Assemble STATE EXPLANATION

Diagram One

REASONINGS OF GREATER TEXTURE

The final part of the system assembles the explanation. As should be clear from the inclusion of causal paths some modification of the basic representation of knowledge in the system is needed. The pattern of reasoning proposed by Toulmin, Reike and Janik can be understood as providing a pattern that extends the rule based paradigm to provide reasoning of greater 'texture.' [9, 3] In order to use the TRJ model for explanation within the expert system paradigm the basic parts of the model will be interpreted as follows: the grounds are the claims held in working memory, the warrants are the rules, the backings are the support for rules, the rebuttals are a set of rules for alternative outcomes, and the claim is the parameter to be modified. Diagram 2 illustrates this interpretation.



The additional resources of the TRJ model provide an effective way to assemble an explanation. If it is allowed that the rules of the expert system are an operationalized correlate of claims in a model of the system, then one set of backings will provide the details of the model in terms of causal paths. Further, if the expert system allows the creation of instances of types of objects, it should be relatively easy to isolate the type of the object in the consequent of the rule. Moreover, if multiple backings are allowed, another set of backings could establish why a particular rule has been formulated in terms of particular illustrative cases.

The various explanations are assembled using the backings and rebuttals. An ES explanation indicates the rule being used along with its ES backing. Its form is, 'Rule XXX was used to infer P because <backing>'. This basic form is expanded in the case of instance explanation by determining the differences between the conditions of the two objects. Its form is, 'Part-XXX is in state S and part-xxx is not in that state because <differences in properties>'. The state explanations use CP backings to trace through the causal path to a point where an object deviates from its normal state and an internal cause is found. Its form would be 'Part-XXX is in state S because part-xxx entered state s and the path P links Part-XXX to part-xxx'. If the other item of the IT provides an exclusive contrast, then the explanation is finished. If, however, the contrast is inclusive and the other item in IT is in the consequent of the rebuttal, then the form 'and not Q because <rebuttal>' is added.

LINKS TO OTHER MECHANISMS

The proposed simple model of explanation allows for a variety of explanation types, and these types provide links to the efforts of other researchers.

Although Schank examined common sense accounts of explanation, rather than the narrower field of scientific explanation focused upon here, his comments on cognitive understanding are helpful. [5] He suggests that for cognitive understanding, "the program must be able to explain why it came to the conclusions it did, what hypotheses it rejected and why, how previous experiences influenced it to come up with its hypotheses and so on." [p. 15] This notion of cognitive understanding, however, is capable of two related, but distinct, readings. The first reading focuses upon how the program, as a program, came to a conclusion. In this sense the program must be able to explain the steps that it took. This notion seems to be captured in the idea of strategic explanation advanced by Schulman and Hayes-Roth. [6] They consider strategic explanations to be descriptions of the strategic plans and decisions that determine the system's actions. The second reading focuses upon why the program came to the conclusion it did, given the hypotheses (theories) and evidence represented in it. Suthers' examination of the view appropriate to the expert seems to capture this reading. [7] He suggests that experts would expect programs to give summaries of case evaluations in the fields terminology supplemented with accounts of its reasoning and use of evidence.

The two readings of Schank's account of cognitive understanding are complementary and not competitive. The proposed simple model of explanation indicates a way in which the strengths of each can be combined to produce a robust framework. [4] Strategic explanations provide the detail needed to construct corrective and ES explanations, and the views appropriate to the expert indicate the mechanisms required to construct instance and state explanations.

CONCLUSION

A preliminary prototype of a simple explanation system was constructed by Blake Ragsdell (University of Louisville) and Lisa Wurzelbacher (Thomas More College). Although the system, based on the idea of storytelling, did not incorporate all of the principles of simple explanation, it did demonstrate the potential of the approach. The system incorporated a hypertext system, an inference engine, and facilities for constructing contrast type explanations.

The continued development of such a system should prove to be valuable. By extending the resources of the expert system paradigm, the knowledge engineer is not forced to learn a new set of skills, and the domain knowledge already acquired by him is not lost. Further, both the beginning user and the more advanced user can be accommodated. For the beginning user, corrective explanations and ES explanations provide facilities for more clearly understanding the way in which the system is functioning. For the more advanced user, the instance and state explanations allow him to focus on the issues at hand.

The simple model of explanation attempts to exploit and show how the why? and how? facilities of the expert system paradigm can be extended by attending to the pragmatics of explanation and adding 'texture' to the ordinary pattern of reasoning in a rule based system.

* An earlier version of this paper was presented at the AAAI Workshop on Explanation (August 1988). I would like to thank the members of the workshop for helpful comments and criticisms.

References

[1] A. Garfinkle, *Forms of Explanation*. New Haven: Yale University Press.

[2] C. Hempel, *Aspects of Scientific Explanation*. New York: The FreePress.

[3] D. Rochowiak, "Expertise and reasoning with possibility" in Proceedings of the Second NASA Conference on Artificial Intelligence for Space Applications (Huntsville, AL).

[4] D. Rochowiak, "Extensibility and completeness: an essay on scientific reasoning.' *The Journal of Speculative Philosophy*, Vol. 2 No.4.

[5] R. Schank, *Explanation Patterns*. Hillsdale, N.J.: Lawrence Erlbaum.

[6] R. Schulman and B. Hayes-Roth, "Plan-based construction of strategic explanations." Knowledge Systems Laboratory Report No. KSL 88- 23; Stanford University.

[7] D. Suthers. "Providing multiple views of reasoning for explanation." Forthcoming in the proceedings of the International Conference on Intelligent Tutoring Systems.

[8] M. Scriven, "Explanations, predictions, and laws" in *Scientific Explanation, Space and Time* (H. Fiegel and G. Maxwell, eds.). Minneapolis: The University of Minnesota Press.

[9] S. Toulmin, R. Rieke and A. Janik, *An Introduction to Reasoning*. New York: Macmillan.

[10] B. van Fraassen, *The Scientific Image*. Oxford: The Clarendon Press.

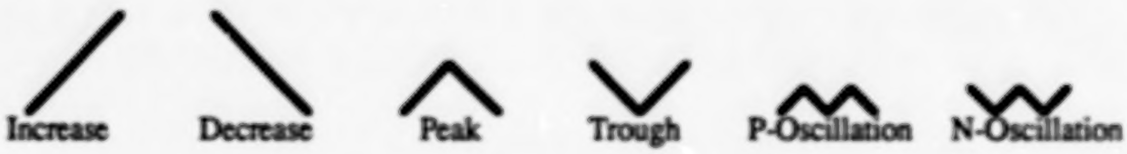


Figure 1. Predefined Prototype Patterns

3- The Representation of Patterns in NNs

3.a- The Network Model

The network model considered here is a recurrent network of discrete value elements. The network is fully connected, with its connection strengths maintained in an n x n matrix *A*, where A_{ij} identifies the strength of the synaptic connection from neuron *j* to *i*. The network is updated synchronously, i.e. every neuron is updated on each cycle. Initially, the activity of the nodes is a real number obtained from the telemetry data or synthetic training data.

The subsequent activities of each node are computed in two steps. First, a weighted sum of nodes inputs is computed to give a 'post-synaptic potential' (PSP), $s_i = \Sigma_j A_{ij} y_j$. The PSP can be either a positive or a negative number, whose magnitude is compared to the neuron's threshold, Q_i. The activity, x_i, is given by

$$x_i = \begin{cases} 1 * \operatorname{sgn}(s_i) & \text{if } |s_i| \geq Q_i \\ 0 & \text{otherwise} \end{cases}$$

These three-valued neurons allow a much broader set of outputs than binary neurons.

3.b- Placement of Category Prototypes

To create a connectivity matrix with particular fixed points, the outer product of each prototype vector with itself is computed, and the resulting matrices are summed over all prototypes. This process guaranties that the prototype vectors correspond to stable states, if they are mutually orthogonal [6]. For the telemetry problem six prototype vectors are placed in an eight dimensional space. The prototype vectors are made up of ±1 values to place them on the outer boundary of activity space. In general, the outer product procedure leads to large basins of attraction around each of the prototypes. There also tend to be a few spurious fixed points with this approach.

3.c- The Learning Mechanism

The behavior of the NN may be modified by altering its connection strength matrix. The performance of the system can be improved by modifying connections so that the difference between the observed and the desired output are reduced. A variety of learning algorithm which achieves this type of improvement is the Widrow-Hoff or Delta rule algorithm [9].

Delta-rule learning calculates a delta from the difference between the final state reached and the final state desired. Each connection coming into a neuron then has its strength modified by an amount equal to the product of delta and the presynaptic activity. Note that the final result of learning depends on both the learning rate and the order in which the input/desired-output pairs are presented.

c-5

4- Training Design

Our purpose in applying a training sequence to the telemetry network is to change the shape of its basins of attraction so that the correct classification is achieved.

4.a- Approach

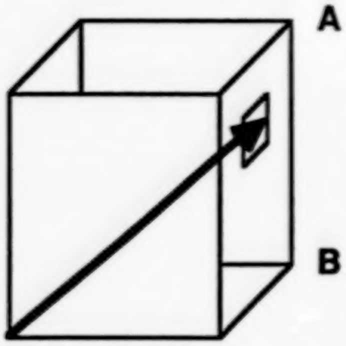
Two subsets patterns makeup a full training set associated with each prototype vector or category: a set of 'good' patterns that lies on the desired boundary of attraction, and a set of 'bad' patterns that lies a little beyond the the boundary. For simplicity, both the good and bad patterns were placed on a hypersphere. The range of radii used for the bad pattern hypersphere was 1-2 times the radius of the desired basin. To obtain a point on a hypersphere around a prototype, a random number is added to each component of the prototype vector, such that the Euclidean distance between the prototype and the constructed point is equal to the desired radius. The random numbers for each component are chosen to lie between 0 and the portion of the radius not yet accounted for. An example of applying this procedure is given below for a desired basin radius of 0.7:

Prototype Vector ---> Training Vector with total noise applied = 0.7

(1 1 -1 -1 1 1 -1 -1) ---> (1.475 1.006 -.897 -.858 1.018 0.792 -.569 -.935)

In order to apply the Delta rule, there must be a desired final state associated with every training vector. For the good training vectors, the desired final state is the prototype. However, for the bad training vectors, a reasonable desired state must be chosen. The method we chose for selecting the desired final state for the bad vectors was to use the corner closest to the input state. This approach worked better than the other two approaches tested: all unclassified vectors were sent to the origin, or to the compliment of the prototype nearest to the input (If the nearest prototype is (-1,1,1,-1), the desired output would be (1,-1,-1,1)). Training on bad vectors tended to destabilize the network when either the origin or the compliment were used as the desired state. This seemed to to occur, because to reach these desired states the trajectory often had to cross the basin of attraction of other category.

The desired final state of a bad pattern is calculated in the following way:
Find k such that $(|t_k| - |x_{0k}|) = \max(|t_i| - |x_{0i}|)$.
Reverse the sign of bit k of the closest prototype to obtain a desired vector that is Hamming distance one from the prototype.
Fig. 2 shows a bad input vector in R^3 and its respective prototype and desired final states.



The input vector C to be rejected is derived from the vector to corner A, and has desired final state of the next closest corner B.

Figure 2. Selecting the Desired Final State for Rejected Vectors

5- Testing Network Performance

To study the effects of training, training data for three different basins of attraction were studied using several different learning rates. For each set of parameters, the final matrix of connection strengths was stored for performance testing.

The connection matrices from the various learning sessions were tested by applying a set of test input vectors. The test data were chosen at random, and were constrained to lie at particular distances from the prototype vectors. Each trained network is scored on the number of correct classifications it achieves on the test data. The inputs are created to provide two different methods of estimating the attractive regions of a network. The first method uses test vectors whose distance from the prototype is normally distributed. One can then count the number of correct classification of a large number input points. The data shown in Fig. 3 agrees with the intuitive notion that training data lying on a larger hypercube around a prototype vector leads to a larger basin of attraction. It illustrates that the fraction of points converging to the prototype does increase as the size of the trained radius increases.

The second method measures the number of classifications made at randomly selected point lying on a sphere at a particular distance (eg. from 0.3 to 0.9 at intervals of .2), and demonstrates the fall off of attractive strength with distance. If a spherical basin of attraction is created by the training procedure then any points outside of the desired radius would not be classified and all of those within the radius would be classified. Fig.4 clearly shows that this does not happen. Instead one finds a significant drop in the percentage of test patterns classified as the radius of the test patterns falls below the desired radius. This implies that the resulting basin of attraction is not spherical, but that on the average it is approximately the right size.

The rates for both connections and thresholds have a significant effect on the performance of these networks. The threshold learning rate seemed to have a destabilizing effect as shown in Fig. 5. The majority of the patterns end up converging on the origin. Learning rates for connections have an optimal range in which learning can take place.

6- Discussion

There are several directions in which this work should be expanded. First, one needs to try more sophisticated learning algorithms than the Delta rule. The Delta rule involves the difference of two terms. The first is essentially an outer product of the input and the desired output. The second is an outer product of the input and the actual output. However, the distance to where a trajectory actually ends up has more to do with where the nearby prototypes are rather than how large a change in the connection matrix must be made. One alternative is to scale the product of input and desired output by the following two factors: the initial rate of change away from the correct prototype, and divide by the difference between the input and the desired output.

Another direction of improvement involves making use of context information. The desired sensitivity of the telemetry classifier varies with time of day, season, and spacecraft activity. Thus, a more sophisticated NN pattern classifier would be able to select the desired sensitivity from input data. One approach is to construct a pair of NNs where the first NN identifies the sensitivity category based on context information, and use this information to set the parameters of the second NN which would actually classify the telemetry data.

In its current form, the network can provide useful information to an expert system. A satellite diagnostic system, for example, could identify a telemetry point as showing a steady increase if the network identifies this behavior several times in a row. Thus, repetition could partially overcome the fuzziness of the basin of attraction boundaries.

Although some ability to control basins of attraction has been demonstrated, the refinement in constructing decision surfaces through learning found in some feedforward networks [8] is a long way off. However, the pursuit of classification in recurrent networks remains an important goal. The brain makes use of processes running on several different time scales. For example, edge detection, allocation of attention, and learning are examples of processes whose temporal scale are at least an order of magnitude apart [2]. If NNs are to be used in producing cognitive capabilities, then the ability to use processes at different temporal scales is critical. At present, fixed points provide the only way of this type of communication.

Acknowledgements

This work was done in part at the Mathematics and Computer Science Department, Drexel University as an independent study project.

References

1- Anderson, J.A., Silverstein, J.W., Ritz, S.A., Jones, R.S.: Distinctive features, categorical perception, and probability learning: Some applications of a neural model. Psycho. Rev. 84:413-51 (1977).
2- Eilbert, J.L., Guez, A.: Attentional states and behavior modes in a hierarchical neural network. Proceedings First IEEE Conference on Neural Networks June 20, 1987.
3- Feldman, J.A., Ballard, D..H.: Connectionist models and their properties, Cognitive Science, 6:205-254.
4- Guez, A., Protopopescu, V., Barhen, J.: On the stability, storage capacity and design of nonlinear continuous neural networks, (to appear in IEEE Man, Systems, & Cybernetics).
5- Hinton, G.E., Anderson, J.A.: Parallel Models of Associative Memory, Hillsdale,N.J.: Lawerence Erlbaum Ass. 1981.
6- Hopfield, J.J.: Neural networks and physical systems with emergent collective computational abilities. Proc. Natl.Acad. Sci. (USA) 79:2554-2558 (1982)
7- Peronnaz, L., Guyon, I., Dreyfus, G.: Collective computatinal properties of neural networks: New learning mechanisms. Phy. Rev. A. 34(5):4217-4229 (1986).

8- Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. In: Parallel Distributed Processing v.1. Rumelhart, D.E., McCelland, J.L. (eds.). Cambrdige, MA: MIT Press. 1986.
9- Stone, G.O.: An analysis of of the Delta rule and the learning of statistical associations. In: Parallel Distributed Processing v.1. Rumelhart, D.E., McCelland, J.L. (eds.). Cambrdige, MA: MIT Press. 1986.

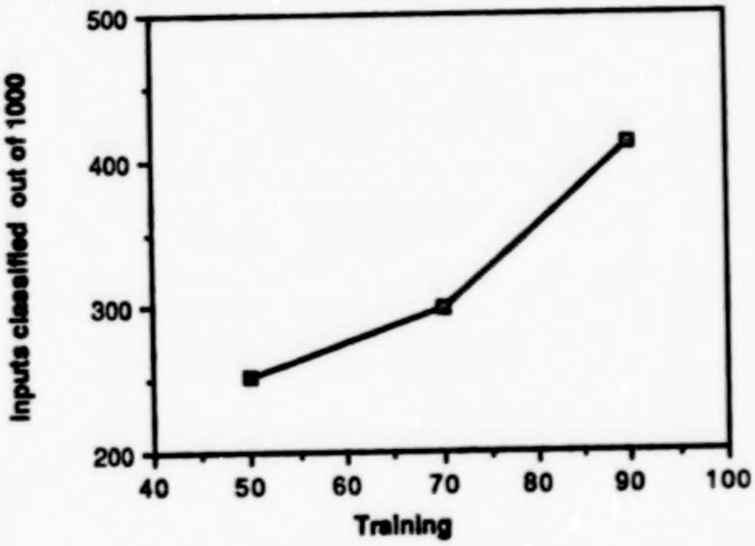


Figure 2. Effects of Training on Basin Size

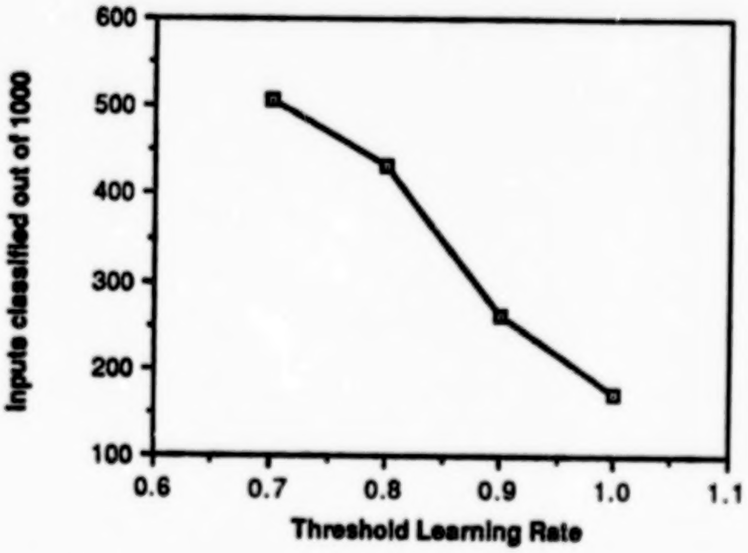


Figure 5. Effect of Threshold Learning Rate

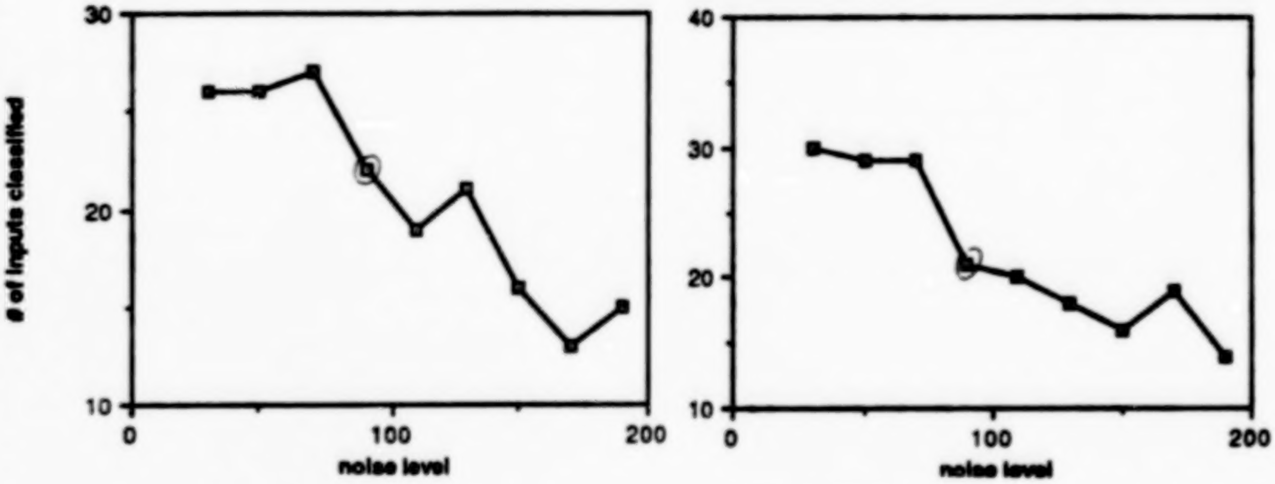


Figure 4a.
Basin Size

Figure 4b.

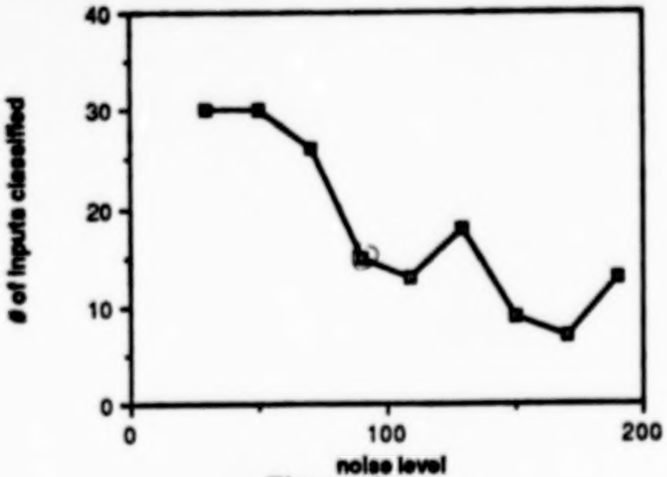


Figure 4c.

Figure 4. Basin Behavior of 3 Networks with
Input on Increasing Spherical Boundaries

N89-15596

An Expert System for Satellite and Instrument Data Anomaly and Fault Isolation

Carl Busse, Jet Propulsion Laboratory
California Institute of Technology

A prototype Generic Payload Operations Control System (GPOCC) is being developed at the NASA Jet Propulsion Laboratory to provide a low-cost command and control processing center for science instruments and small payloads. The GPOCC supports the difficult transition from integration and test to flight operations. The prototype will incorporate four expert systems to perform telemetry, command, and mission planning functions as well as telecommunications scheduling. The first of these expert systems to be developed will perform telemetry data analysis and fault isolation, as well as propose corrective action.

This Data Analysis Module (DAM) will monitor telemetry data and perform continual data monitoring and trend analysis based on a knowledge base and historic data archived on an optical disk storage device. The system maintains a continuous "knowledge" database of past system performance characteristics.

The Data Analysis Module will be partitioned into four stages:

MONITORING - monitoring and interpreting instrument and satellite behavior.

DIAGNOSIS - determine origin of system malfunctions inferred from knowledge base.

PREDICTION - inference of predicted performance based on historic performance and current trends.

RECOMMENDATION - developing and prescribing corrective action for diagnosed problems.

The goal of the Data Analysis Module is to achieve consistent, dependable and validatable performance, to demonstrate thorough, reliable and fast reasoning, and to reduce the concentration demanded of flight analysis personnel.

The work described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

Introduction

The Data Analysis Module (DAM) is a project of the Institutional Computing and Mission Operations division of the California Institute of Technology, Jet Propulsion Laboratory. The Data Analysis Module is an expert system element intended for future incorporation in a purposed Generic Payload Operations Control Center (GPOCC). The purpose of the DAM project is the design and development of an expert system to monitor spacecraft and science instrument telemetry data, isolate and diagnose faults, pinpoint the probable fault location and recommend corrective action.

Background

The Jet Propulsion Laboratory is a lead center for NASA's program of planetary exploration and earth science. In support of this role JPL has pursued areas of technology associated with the advancement of the spacecraft and science instrument operations environment. To concentrate emphasis efforts in new areas of technology the Laboratory established the Office of Space Science and Instruments (OSSI) and the Office of Technology and Applications Programs (TAP). *[JPL Annual Report 1988]*

The concentration of the Laboratory on technology in the mission operations domain has centered on automation of instrument and spacecraft command and control functions. Automation in the mission environment is significant for three reasons. First, carefully selected automation reduces the workload of the operations personnel. Second, automated and verifiable computer-based tools *[Harmod 1987]* improve the accuracy of data processing and assist space flight and instrument control engineers in monitoring of spacecraft and mission sensors where operating data rates may greatly exceed the ability of individuals to monitor successfully. And third, as the number of missions increases, the number of trained and experienced flight support personnel cannot keep up with the extreme demands caused by information overload. Automated aids and operator assistance allow for productivity enhancement and maintaining the required level of flight support. *[Hansen 1988]*

To automate the flight and instrument integration and control process, the Laboratory has directed a major effort toward the incorporation of artificial intelligence into spacecraft sensor and space vehicle integration and test, and flight operations areas.

This effort has led to the development of several knowledge-based systems to improve the JPL spacecraft and instrument command and control process. Specifically these expert systems are:
SHARP, the Spacecraft Health Automated Reasoning Prototype, is one

of a number of Mission Operations Productivity Enhancement Program (MOPEP) activities. SME, Spacecraft Monitoring Environment, is being incorporated into the satellite integration and test domain. EPDM, the Electrical Power Data Monitor, is being designed to monitor the Voyager spacecraft power systems, and DAM, Data Analysis Module.

The Spacecraft Health Automated Reasoning Prototype (SHARP)

The Spacecraft Health Automated Reasoning Prototype has been designed to incorporate the experience of the lead Voyager spacecraft telecommunications engineer into a useable knowledge base. Data from this knowledge source is assimilated in a knowledge base that will be used as an around-the-clock mission operations assistant in support of the Voyager spacecraft's upcoming Neptune Encounter. The LISP expert system shell was used in the development. Data Views provides the graphical interface. SHARP is implemented on a Symbolics 3670.

Spacecraft Monitoring Environment (SME)

The Spacecraft Monitoring Environment is being developed to aid in the Galileo spacecraft integration process. The SME will provide a real time autonomous spacecraft test sequencing and data monitoring of integration and test activity. SME resides on a Sun 386/i. Data Views is being used to provide high-level contextual graphic displays and windowing capability.

Electrical Power Data Monitor (EPDM)

The Electrical Power Data Monitor is currently under conceptual design. The EPDM will be designed to oversee the Voyager spacecraft power systems during the Neptune Encounter. The C Language Integrated Production Systems (CLIPS) developed by Johnson Space Center Mission Planning and Analysis Division's Artificial Intelligence Section, will be used as the expert system shell. The EPDM development will be on a Sun 3/260.

Generic Payload Operations Control Center (GPOCC)

The Generic Payload Operations Control Center is a concept being developed to apply automation to instrument and satellite testing, as well as a mission operations environment. The GPOCC will couple expert systems with high level contextual graphical data displays for ease of user interpretation. A modest prototype was developed on a Apple MacIntosh II to demonstrate user interfaces and functionality of the GPOCC concept.

The Generic POCC will apply expert system technology to four areas: 1) mission flight planning for science instrument and flight sequence planning, and command constraint checking; 2) control of on-board data storage devices, memory management, and memory comparison; 3) DSN 26-meter subnet and TDRSS telecommunications link scheduling; 4) satellite telemetry data monitoring, trend Analysis, prediction forecast, anomaly detection, fault isolation, diagnosis, and corrective action strategy. *[JPL D-5435 1988]*

The Generic Payload Operations Control Center effort is intended to support the arduous transitions between development, instrument and satellite integration and test, and flight operations.

However, lack of a funding source and development manpower restrictions has permitted only piecemeal development. For this reason, an implementation strategy was generated which provided for a phased progression of development. The Data Analysis Module was selected for initial implementation. DAM builds on the experience gained in development of telecommunications and power systems, and spacecraft integration and test systems. However, these systems were designed to be operated within a restricted domain of their own unique environment (i.e. telecommunications, powers systems). DAM is intended to develop an expert system that encompassed operation of all the subsystems of a complete spacecraft.

Other NASA Center Expert System Implementations

Several other NASA space centers have developed expert systems for use in a mission operations environment. All of the systems have been well thought-out and impressively implemented. The Goddard Space Flight Center has developed two such systems. The Communications Link Expert System (CLEAR) for Cosmic Background Explorer (COBE) developed by the GSFC Data Systems Applications Branch of the Data Systems Technology Division, *[Hughes 1987]* and MOORE (named after Mr. Bob Moore at TRW who was the knowledge source) a prototype expert system for diagnosing TDRSS spacecraft attitude control problems, developed by Westinghouse Electric for the GSFC. *[Howlin 1988]*

The Johnson Space Center Mission Planning and Analysis Division has developed an expert system called INCO, for the Instrumentation and Communications Officer prototype designed to monitor and analyze Space Shuttle telecommunications links. *[Madison 1988]*

Data Analysis Module

The Data Analysis Module will monitor telemetry data and perform continual data monitoring and trend analysis based on its knowledge base and historic data archived on an optical disk storage device. Because of cost constraints, an off-the-shelf hardware PCM decommutator will be used to frame synchronize and channelized the data stream instead of building a software frame synchronizer. The DAM prototype will include a worm-drive optical disk to maintain a continuous "knowledge" database of past system performance characteristics.

The Data Analysis Module contains four rule domains. These domains are:

MONITORING - monitoring and interpreting instrument and satellite behavior.

DIAGNOSIS - determine origin of system malfunctions inferred from knowledge base.

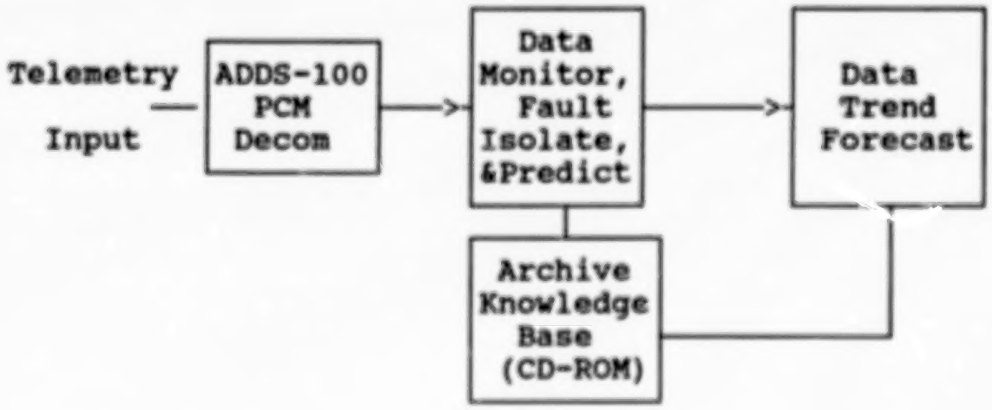
PREDICTION - inference of predicted performance based on historic performance and current trends.

RECOMMENDATION - developing and prescribing corrective action for diagnosed problems.

The goal of the Data Analysis Module is to achieve consistent, dependable and validatable performance, *[Stagner 1988]* to demonstrate thorough, reliable and fast reasoning, and to reduce sometimes-overwhelming loads placed on flight analysis personnel.

The DAM processing flow is show in figure 1.

Figure 1 Data Analysis Module Process Flow


Telemetry Input
ADDS-100 PCM Decom
Data Monitor, Fault Isolate, &Predict
Data Trend Forecast
Archive Knowledge Base (CD-ROM)

Data Analysis Module Mission Domain

In implementing the Data Analysis Module, the science and mission user domain framed the overall structure and requirements of the DAM. DAM will function in an instrument test environment, a spacecraft integration and test environment, and lastly, in an actual mission operations environment.

In the instrument test environment the DAM will allow the instrument principal investigator or scientist to observe operating conditions during development in the same environment and equipment configuration as will be used in flight.

In the spacecraft integration and test environment the DAM must provide data-handling flexibility and integrated displays to lend itself to the task of releasing satellite integration and test personnel from constantly modifying a complex data handling system as system requirements change.

The DAM must support a mission operations environment which will be a natural progression from the instrument and satellite integration and test environment. The DAM support of the Mission Operations System (MOS) will differ little from the prelaunch Ground Data Systems testing and MOS training activities. The major difference is the addition of "live" flight telemetry data input to the input process. This similarity significantly aids in the transition from single-instrument integration to full-up on-orbit operations.

The DAM telemetry task will be coupled to a display systems design to provide more user-efficient data interpretation by use of symbolic representation including pop-up windows, scroll bars, graphics, colors representative of data states, and interactive icons.

Data Analysis Module Design Concept

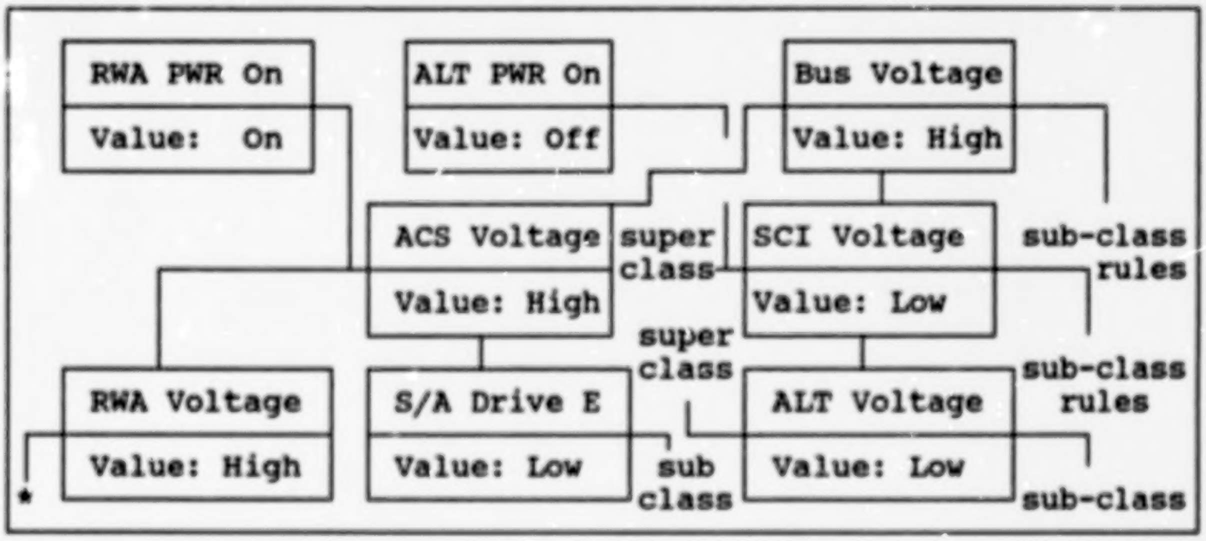
Knowledge representation is a central issue in the development of intelligent systems. While a number of knowledge representation formalities and techniques have been developed, most are based on semantic networks and frames. *[Barr 1981;Cho 1985]* These may have limited application domains which depend on complex deductive strategy.

Expert systems have been developed to provide telemetry data analysis. These systems have examined data related to a unique spacecraft subsystem (i.e. telecommunications, and power in SHARP and EPDM, respectively). The DAM examines the input telemetry stream on a criticality prioritized basis.

Early in system design each telemetry channel was assigned a criticality level. Once this was done, a relationship for each parameter was established. The level two relationship is flowed down through to sub-tier prioritized classes. The result was an interwoven tree of rule classes based on individual channel priority and importance. This entity-relation approach allows critical spacecraft and sensor faults to be monitored constantly and faults to be quickly detected.

Each prioritized telemetry channel is assigned a frame slot. These slots explicitly point downward to assigned subclasses, and up to super classes. This enables DAM frames (a knowledge representation scheme that associates features with an object in terms of various slots and slot values) to capture hierarchical taxonomies. An example of the DAM rule hierarchy is shown in figure 2.

Figure 2 DAM Rule Classes



Hardware Configuration

To reduce the cost of the initial implementation the conceptual testbed was established on an IBM PC/AT clone using CLIPS as the expert system shell. Simulated spacecraft telemetry was used for the data source. Formal prototyping will be done in the JPL Mission Operations Division's Operations Engineering Laboratory (OEL) on a SUN 4/260 using an Advanced Digital Data Systems ADDS-100. PCM decommutator to provide conditioning of the serial telemetry data stream.

The initial testbed configuration is:

System 1800 AT 80286 running at 8 MHZ
80287 co-processor
640 Kbyte RAM
2.5 Mbyte Extended RAM
with 80 Mbyte hard disk and 30 Mbyte auxiliary storage
1.2 Mbyte floppy disk drive
60 Mbyte streaming tape storage
VGA display with NEC Multisync II color monitor

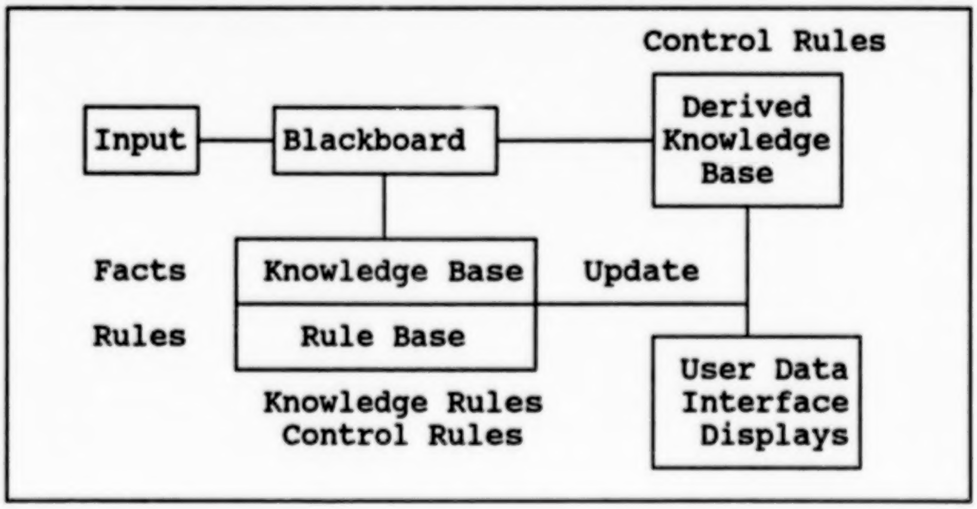
The formal OEL prototype configuration will be:
Sun 4/260 68020 running at 16 MHZ
8 Mbyte RAM
two 71 Mbyte hard disks
60 Mbyte streaming tape storage
1152 x 900 bit-mapped color display

Development Approach

The DAM will be implemented in two stages, a conceptual testbed, and a formal prototype. To validate the conceptual design of the Data Analysis Module a testbed consisting of approximately twenty five rules for the five spacecraft subsystems. Following the testbed demonstration a formal prototype will be implemented. The prototype will consist of roughly one hundred twenty five rules and will be configured as a flight support workstation.

The DAM testbed will be use to develop the initial telemetry channel hierarchial relations. The testbed will demonstrate our understanding of the telemetry problem domain. This initial testbed model will be a proof of concept. Once the testbed has shown our comprehension of the domain, prototype design will begin. Testbed development is expected to be completed by the end of the year. The conceptual design of the testbed is shown in figure 3.

Figure 3 DAM Testbed Conceptual Design



Both the testbed and the formal prototype will use CLIPS as the expert system shell. CLIPS was chosen for several key features: these include forward rule chaining, rule syntax, multi-field functions, embeddable in C programs, window and mouse interfaces, portability (testbed to prototype), tool set (rule break pointing, fact address comparisons, style checking and cross-referencing), and complete documentation.

The prototype will allow evaluation of the DAM hardware and software performance, as well as ferret out problems employing actual telemetry data sets. The formal prototype development will be completed by the end of 1989.

The monitoring, diagnosis, and recommendation rules are combined into a single task. The predictive model is generated from past and current data activity. The DAM will also log all expert system activity for later analysis.

System Architecture

The basic testbed architecture incorporates three components: the CLIPS expert system shell, the input data source, and the output user displays.

The ADDS-100 telemetry decom provides the DAM with simulated communications links from the Tracking and Data Relay Satellite System (TDRSS), White Sands Ground Terminal (WSGT) and the JPL Deep Space Network (DSN) 26-meter subnet. This telemetry source data is frame synchronized, and decommutated by the PCM hardware decommutator. The channelized data is archived.

The DAM expert system provides telemetry data monitoring and interpreting, and fault isolation. The DAM will then determine origin of system malfunctions inferred from the knowledge base and develop and prescribe corrective actions for the diagnosed problems.

The output user displays inform the user of current status and alert the user to any data irregularities.

Data Monitor

The Data Analysis Module monitors data values to detect faults. These faults are identified, a diagnosis is performed, and corrective action strategy proposed. As the DAM examines the input serial telemetry stream each data channel is placed in a predefined frame slot and compared against initial user defined expected data boundary value conditions. If a value exceeds the established boundary value the rules associated with that data channel fire. If a nominal value is present the current value is compared against previous data values.

Fault Diagnosis and Corrective Action Recommendation

When channel values exceed boundary conditions faults are isolated and corrective action recommendations are predicted by the compound rules classes governed by assertive clauses. For example:

```
(assert   (RWA run-away
           ACS Voltage High
           SCI Voltage Low
           S/A Voltage Low
           RWA PWR     On))
```

Predictive Modeling

While the DAM is performing continual data monitoring trend analysis based on historic data is preformed. Trend variations are examined for fault conditions, inference of predicted performance is made based on historic values and recommendations are made for corrective action. The system maintains a continuous "knowledge" base of past telemetry system performance characteristics.

Knowledge Representation

The DAM man-machine interface will be via easily interpreted contextual graphic displays. These interactive video displays will provide a direct representation of the intrinsic images associated with the instrument and satellite telemetry and telecommunications systems. Multiple display screens with pop-up windows and high resolution graphics will be linked through context and mouse-sensitive icons and text.

Implementation

The DAM design will insure that implementation meets the requirements specified in JPL software standards as well as software system integration and test standards. *[JPL D-4000; D-5000 JPL 1988]*

Summary

Based on the knowledge gained in the design and development of the Data Analysis Module, the Generic Payload Operations Control Center will provide for consistent, dependable and validatable performance. It will demonstrate reliable reasoning, and reduce the requirement for a large integration and test, as well as a mission flight support staff.

Acknowledgments

The author gratefully acknowledges the guidance and suggestions from Mr. Harry Avant, Ms. Irene Wong, Dr. James Willett, Mr. Warren Moore, Mr. Ray Stagner and Mr. David Hermsen of the JPL; Dr. Rogers Saxon and Mr. Neal Kaufman, of CypherMaster; This paper could not be prepared without the editorial assistance of Mr. Robert Embry and Ms. Fleta Gallagher of JPL.

References

1. Barr, A., et al, The Handbook of Artificial Intelligence, Vol.1, Los Altos: William Kaufmen, 1981.

2. Cho, J.W., et al, "KPSP: A Knowledge Programing System Based on Prolog", 4th International Conference of Entity-Relationship, Chicago, 1985.

3. D-4000, JPL Software Management Standards, California Institute of Technology, Jet Propulsion Laboratory, Pasadena, (1988).

4. D-5000, JPL Software Integration and Test Standards, California Institute of Technology, Jet Propulsion Laboratory, Pasadena, (1988).

5. D-5435, Generic Payload Operations Control Center Function Requirements Document, California Institute of Technology, Jet Propulsion Laboratory, Pasadena, (1988).

6. Hansen, E., "Lowering the Cost of Satellite Operations", American Institute of Aeronautics and Astronautics, AIAA-88-0549, (1988).

7. Harmon, P. ed., "Expert System Tools", Expert Systems Strategies, (1987).

8. Howlin,K., "MOORE: A Prototype Expert System for Diagnosing Spacecraft Problems", 1988 Goddard Conference on Space Applications on Artificial Intelligence and Robotics, GSFC, 1987.

9. Hughes, P.M., et al,"CLEAR: Communications Link Expert Assistance Resource", 1987 Goddard Conference on Space Applications on Artificial Intelligence and Robotics, GSFC, 1987.

10. Jet Propulsion Laboratory 1987 Annual Report, California Institute of Technology, Jet Propulsion Laboratory, Pasadena, (1988).

11. Madison, R. M., et al, "INCO: Real Time Expert System Prototype for Shuttle Mission Control", 1988 Space Operations Automation and Robotics Conference, Dayton, 1988.

12. Stagner, J. R. "Toward A Certification Methodology For Expert Systems", California Institute of Technology, Jet Propulsion Laboratory, Pasadena, (1988).

N89-15597

A Multiprocessing Architecture for Real-Time Monitoring

James L. Schmidt
Simon M. Kao
Jackson Y. Read
Scott M. Weitzenkamp
Thomas J. Laffey

Lockheed Artificial Intelligence Center
2710 Sand Hill Road
Menlo Park, CA 94025
(415) 354-5209

Abstract

This talk describes a multitasking architecture for performing real-time monitoring and analysis using knowledge-based problem solving techniques. To handle asynchronous inputs and perform in real time, the system consists of three or more distributed processes which run concurrently and communicate via a message passing scheme. The Data Management Process acquires, compresses, and routes the incoming sensor data to other processes. The Inference Process consists of a high performance inference engine that performs a real-time analysis on the state and health of the physical system. The I/O Process receives sensor data from the Data Management Process and status messages and recommendations from the Inference Process, updates its graphical displays in real time, and acts as the interface to the console operator.

The distributed architecture has been interfaced to an actual spacecraft (NASA's Hubble Space Telescope) and is able to process the incoming telemetry in "real-time" (i.e., several hundred data changes per second). The system is being used in two locations for different purposes: (1) in Sunnyvale, California at the Space Telescope Test Control Center it is used in the preflight testing of the vehicle; and (2) in Greenbelt, Maryland at NASA/Goddard it is being used on an experimental basis in flight operations for health and safety monitoring. This talk will discuss both these applications in detail.

PRECEDING PAGE BLANK NOT FILMED

N89-15598

PI-in-a-box : Intelligent Onboard Assistance for Spaceborne Experiments in Vestibular Physiology.

Silvano Colombano
RECOM Software Inc.
NASA-Ames, Artificial Intelligence Research Branch,

Laurence Young
Massachusetts Inst. of Technology and Stanford University

Nancy Wogrin
Stanford University and Digital Equipment Corp.

Don Rosenthal
NASA-Ames, Artificial Intelligence Research Branch

NASA-Ames, Mail Stop 244-17, Moffett Field CA 94035

Abstract

We are constructing a knowledge-based system that will aid astronauts in the performance of vestibular experiments in two ways: it will provide real-time monitoring and control of signals and it will optimize the quality of the data obtained, by helping the mission specialists and payload specialists make decisions that are normally the province solely of a principal investigator, hence the name PI-in-a-box. An important and desirable side-effect of this tool will be to make the astronauts more productive and better integrated members of the scientific team.

The vestibular experiments are being planned by Prof. Larry Young of MIT, whose team has already performed similar experiments in Spacelab missions SL-1 and D-1, and has experiments planned for SLS-1 and SLS-2. The knowledge-based system development work, performed in collaboration with MIT, Stanford University and the NASA-Ames Research Center, addresses six major related functions: a) signal quality monitoring; b) fault diagnosis; c) signal analysis; d) interesting-case detection; e) experiment replanning; and f) integration of all of these functions within a real-time data acquisition environment. Initial prototyping work has been done in functions a) through d).

Introduction

The conduct of experimental science in existing spaceborne laboratories such as Spacelab or future ones such as the Space Station is severely constrained in several respects. The principal investigator (PI) is normally not on the

PRECEDING PAGE BLANK NOT FILMED

spacecraft and communication with the ground is either limited in bandwidth or availability. In any event, because of the open nature of the air-to-ground voice links, free discussion of experimental alternatives is inhibited. Furthermore, the experiment-specific decision-making ability of the astronauts is limited by the training they have been able to receive before the flight and by the time they have available in flight. Longer mission durations make it more likely that contingencies will arise for which the astronaut will have had inadequate preparation.

Considering the limited opportunities that exist for flight experiments, and the scarcity of both space and crew time, an intelligent system to assist in the conduct of spaceborne science seems like a potentially useful tool, especially if it contains much of the experiment specific knowledge known to the PI. At a minimum it would save crew time, but what we are especially targeting is the kind of improvement in the quality of the experiment that comes from a deeper understanding of the processes involved, and from the ability to make timely decisions that can change the course of the experiment on the basis of "interesting" data. In some sense, the present necessity of having the entire experiment pre-planned is likely to lessen the possibility of surprising discoveries, and the PI on the ground is often left with the frustration of not having had a chance to look again at some unusual event, or of mistakenly recording meaningless data, and of having to wait for years for a second chance.

The work we have initiated addresses the following major functions: signal quality monitoring, fault diagnosis, signal analysis, interesting-case detection, experiment replanning and, finally, coordination of these functions by a protocol management subsystem. Our initial prototyping effort has concentrated mainly on the areas of signal quality monitoring, fault detection and interesting-case detection.

In this paper we discuss the operational environment, how artificial intelligence technology can contribute to the attainment of better scientific data, and the experience we have gained during our initial prototyping effort.

The Operational Environment - How a Typical Experiment is Performed -

PI-in-a-box (symbolically [PI]) is initially designed to be used in the context of a series of experiments planned to test theories on adaptation to weightlessness and space motion sickness. The hypothesis tested and confirmed so far by Young et al. [3] is that, during adaptation to weightlessness and readaptation to one-g, signals from the inner ear are reinterpreted by the brain. In brief, on earth the presence of gravity is constantly sensed in the inner ear, and signals from the inner ear sensors (utricular otolith afferent signals) are combined in the brain with [illegible], visual and tactile clues to provide us with a sense of orientation within our environment. In the absence of gravity the inner ear signals are still sent, but, after some adaptation, they are reinterpreted and combined with other environmental clues in ways that are different from those observed in a normal one-g environment. An analogous process of readaptation occurs after returning to one-g.

In order to test the sensory reinterpretation hypothesis it is necessary to perform the same experiments under three different sets of environmental conditions: on the ground before flight, in flight with microgravity and, again, on the ground after the end of the mission. The first ground experiments provide a baseline, the flight experiments show the adaptation that occurs in the microgravity environment and, finally, post-flight ground experiments show the re-adaptation that must take place after gravity returns to play its usual role in the vestibular system.

The experiments are typically performed by teams of two people, who take turns being subject and experimenter. Examples of typical experimental setups are a rotating dome, a rotating chair and a sled on rails. The subjects might be asked to look into the dome, or sit in the rotating chair or on the sled while it is being moved. During the experiments, which typically lasts a few minutes, the subjects are asked to perform tasks that indicate their spatial orientation, for example, they may be directed to look at specific areas, describe sensations, or point to a perceived "up" or "down". Eye movements are a very important source of information in many of these experiments and can be recorded via electro-oculography (EOG). This technique is based on the fact that each eye acts as a dipole within the head, and its movements cause electrical field changes that can be detected by electrodes appropriately placed on the skin. Other types of movements, such as iris dilation, can be observed with video cameras and suitable image processing, but this type of observation has not yet been accomplished in flight.

Potential Role of Artificial Intelligence Technology

At present, in his/her mode as an experimenter, the on-board astronaut follows a protocol document in the form of a checklist. This checklist guides the specialist in preparing the instrumentation for the experiment and in instructing the subject on the actions required of him or her. The protocol document also provides a guide for evaluating the quality of the data produced by the experiment, and for what should be done when the quality achieved is not sufficient for meaningful results, or when there is some unexpected occurrence.

In spite of long and rigorous training imparted to mission specialists, the checklist system is inadequate in helping them keep track of events unanticipated in the protocol, and in helping them diagnose complex experimental problems. A consequence of the limitation of the checklist system is that problems can go unnoticed, or the PI on the ground must be summoned to help the specialist diagnose and correct the problem. The likelihood of timely and effective repair is higher when the expertise is available at the problem site - in the spacecraft.

An knowledge-based system can monitor several variables at once, and direct the mission specialist's focus of attention to problem areas. It can also aid the specialist in diagnosing problems of different levels of complexity, within the limits of the knowledge built into the system and the time available.

An additional, potentially extremely valuable contribution of this technology is in detecting opportunities, rather than just detecting problems. By this we mean the ability to detect that a particular data set reveals some unexpected behavior of the system under study and that this discovery may warrant a repetition of the experiment or a change in the set of experiments that is to follow. This is the type of insight that is normally the province solely of the PI. Can we build a system with enough knowledge and, perhaps, modeling ability, to achieve insights of this type? We are not yet ready to answer this question in the affirmative, but we report below on the experience we are beginning to accumulate.

The System Design

Our initial top level system design is illustrated in Figure 1. The hub of the system is the "protocol manager". This subsystem interacts with the astronaut and with the experiment, and serves as a controller for the remaining four major subsystem functions: a) data quality monitoring and fault diagnosis, b) interesting-data detection, c) suggesting new experiments and d) scheduling or rescheduling experiments and resources. Here we report on work done in a) and b).

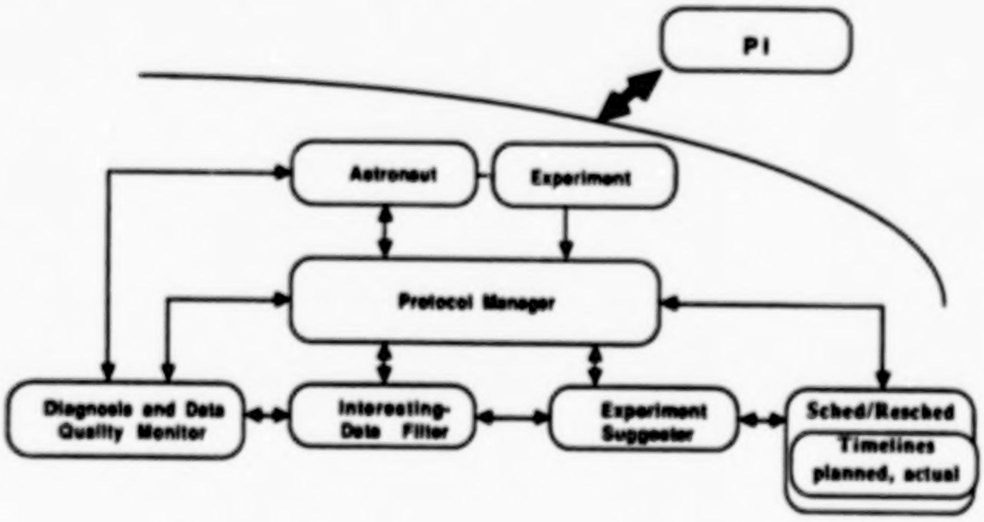


Figure 1: Top level system design of PI-in-a-Box

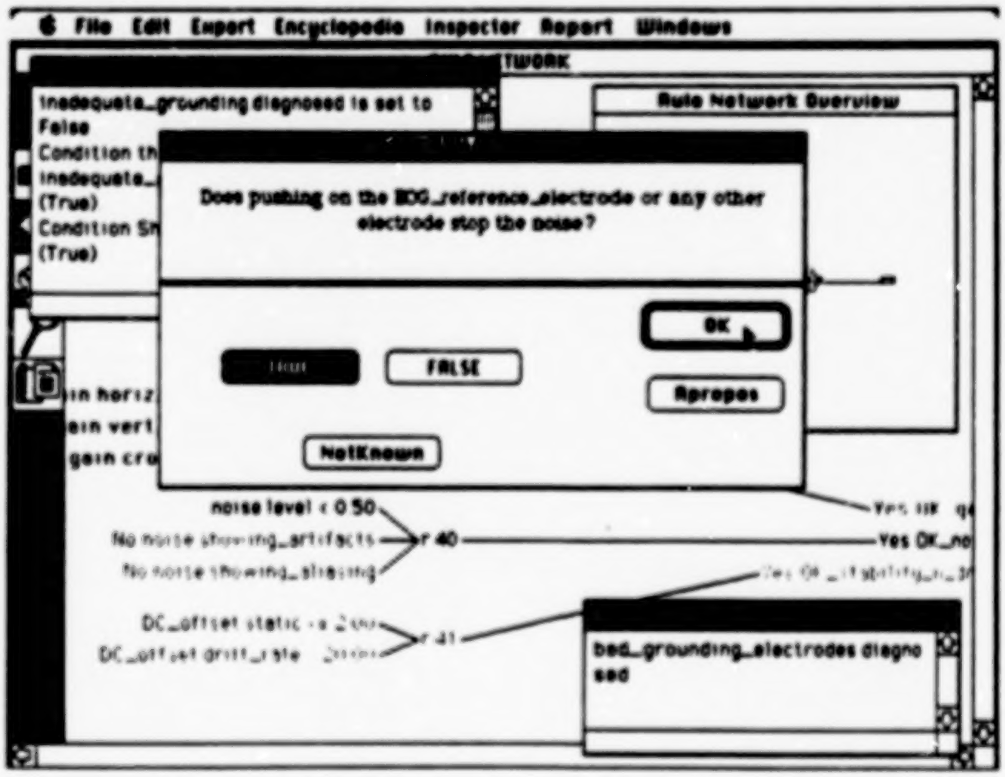
Signal quality monitoring and fault diagnosis

As a first prototype we built a small diagnostic knowledge based system for the EOG eye movement signal. Quality monitoring and diagnosis of this signal is representative of a type of monitoring and diagnostic activity useful for most experiments.

ORIGINAL PAGE IS
OF POOR QUALITY

The system uses a simple backward chaining of rules where an attempt is made to establish that the signal received is "good". In order to establish this hypothesis, the system looks at a number of properties related to signal gain, noise and stability. If any of the values of these properties is not what would be expected for a good signal the system attempts to diagnose the problem and to suggest a solution. When the system detects an unexpected value, but cannot find an explanation and suggest a course of action, it recommends that the PI be contacted. However, the purpose of the system is neither to keep the PI out of the loop, nor to require less of the specialist in terms of understanding and skill. On the contrary we hope that this tool will help make both the PI and the mission specialist more productive and better decision makers by acting as an "intelligent partner".

The current knowledge base used to monitor and diagnose the EOG signal is embodied in about 60 rules at this time. This size has been sufficient to produce a meaningful demonstration, although the user is still required to input data that will be input automatically in the operational system, . Two typical interaction screens and a tree of the types of problems detected so far are shown in Figures 2, 3 and 4 respectively.

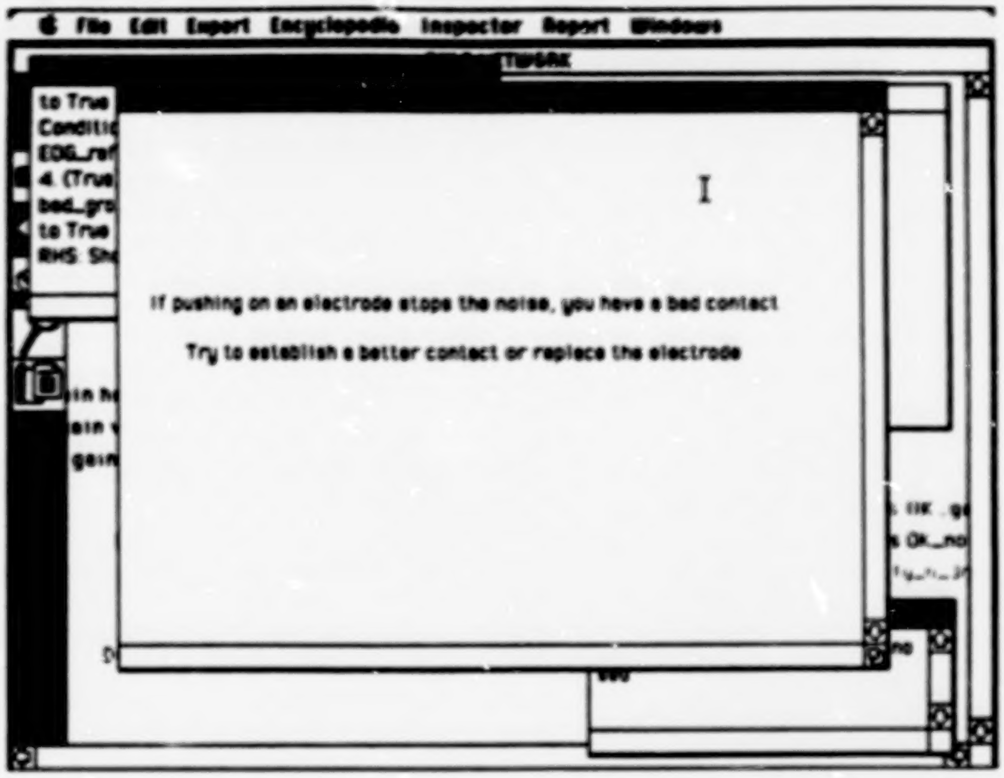


inadequate_grounding diagnosed is set to False
Condition th
inadequate_
(True)
Condition Sh
(True)

Does pushing on the EOG_reference_electrode or any other electrode stop the noise?

bad_grounding_electrodes diagnosed

Figure 2: A prototype screen requesting input from the astronaut



If pushing on an electrode stops the noise, you have a bad contact

Try to establish a better contact or replace the electrode

Figure 3: A prototype screen suggesting a course of action

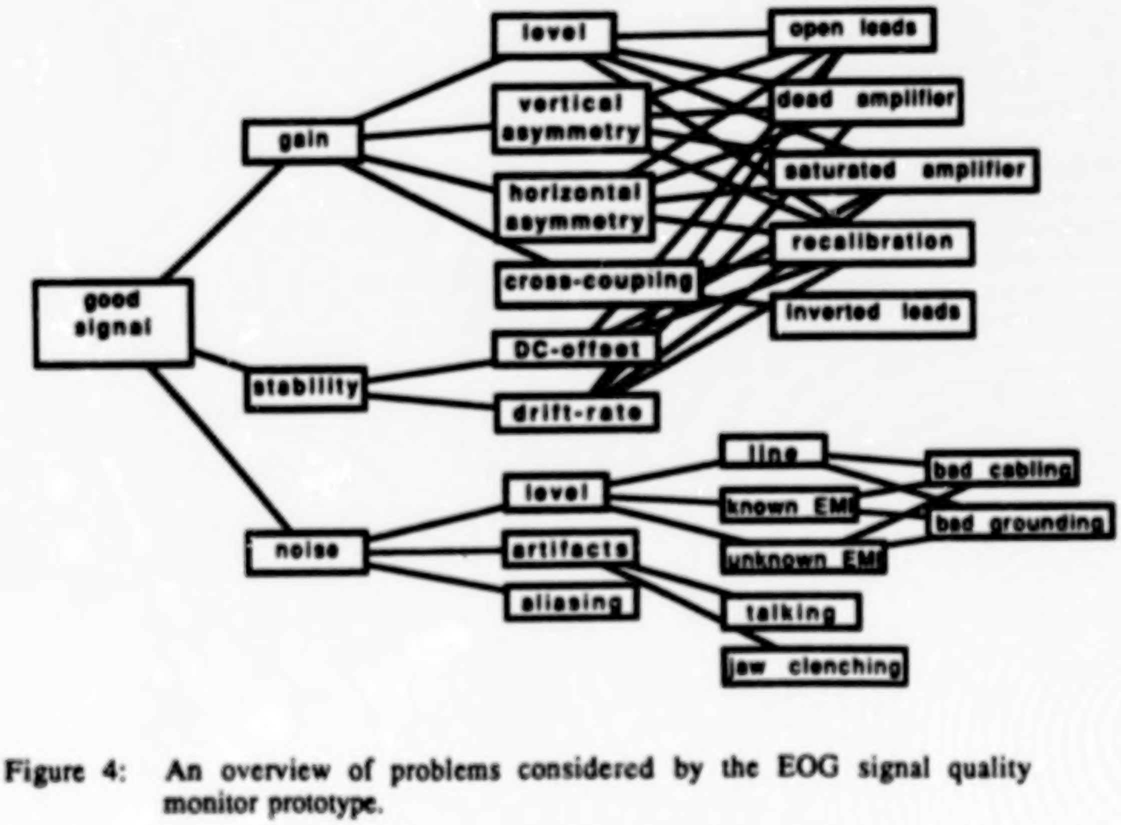


Figure 4: An overview of problems considered by the EOG signal quality monitor prototype.

ORIGINAL PAGE IS
OF POOR QUALITY

Prof. Laurence Young, an internationally recognized expert in vestibular physiology, has served as domain expert in our initial work. We chose the commercial tool Nexpert Object [2] as the system building tool. Our main reason for selecting Nexpert was that it appeared to be the most powerful knowledge based shell that could run on a range of mini-computers and microcomputers, including the IBM PC and the MacIntosh. Its price was within our initial budget, and it provided a powerful user interface that made it suitable for demonstration purposes.

Interesting-case Detection

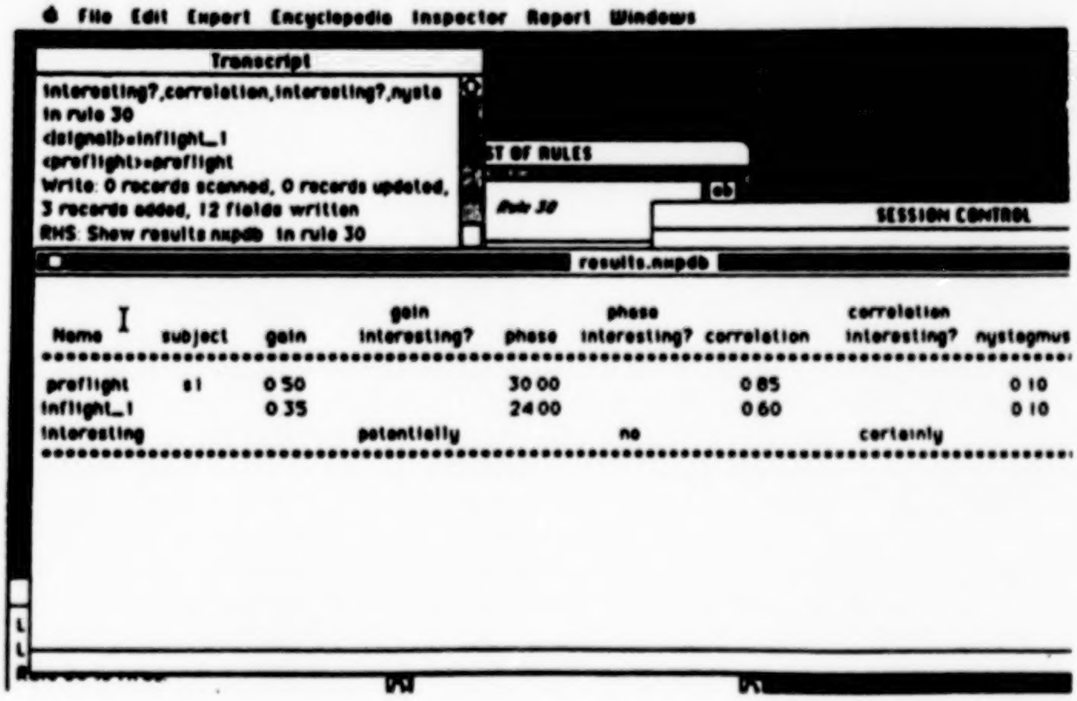
We think of Interesting-case Detection as the ability to distinguish among normal expected good data , experiment artifacts, and unexpected data that is interesting enough to lead to a recommendation for changes in the original experiment plan, either for a second look or for the exploration of previously unexpected possibilities. This capability, which we hope to achieve, is what would set this system apart from normal diagnostic and planning systems, and truly earn it the name PI-in-a-Box.

This portion of our research focused on a vestibular sled experiment. In this experiment the subject sits on a sled which rides back and forth on a track. The subject's eye movement is measured by EOG. Comparison of pre- in- and postflight values of these parameters provides a test of the orientation model. For this phase of the work, the signals were assumed to be of good quality. The decision-making behavior we tried to reproduce was that of a scientist who looks at the data for a pre- and an inflight experiment, compares the two sets, and decides in a manner of minutes, during the experiment, whether the results are interesting enough to warrant further exploration. A consequence of this assessment might be to repeat the experiment, modify the same or subsequent experiments, or simply note the finding and proceed as planned.

Usually a scientist uses a graphical representation of data suited to the necessary interpretation, some examples being graphs of distributions or of derivatives. The scientist identifies attributes of interest by describing qualities of the graphical representation, such as "beats", "phases", or the shape of a curve. Translating raw data into the kinds of attributes a scientist actually thinks about is, in general, a difficult task. For this prototype, we identified the relevant attributes that would have to be extracted from the data, and asked the user for these values. In the future, the values of these attributes will be obtained by traditional signal analysis techniques (e.g. [1]). For this experiment we identified the following parameters: degree of linear nystagmus, and gain, phase and correlation of eye movement in relation to sled motion (Table 1). The values of these parameters can be obtained by algorithmic means, such as Fast Fourier Transforms, peak to peak ratios, or cross correlation techniques. The screen shown in Figure 5. compares these parameters for a pre- and an inflight experiment, and suggests whether the discrepancies detected should be viewed as interesting or not.

Parameter	Obtaining the Value	Value We Use
correlation	Cross correlation technique	Number representing degree of correlation
Inystagmus	Fortran program and rule base	Number representing certainty of presence of nystagmus
gain	Peak to peak ratio of sled motion to eye movement	Number
phase	FFT	Number representing degree of phase shift

Table 1: Parameter values considered in the Interesting-case Detection prototype for the sled experiment.



```
interesting?,correlation,interesting?,nysta
in rule 30
<dsignaldb>=inflight_1
<preflight>=preflight
Write: 0 records scanned, 0 records updated,
3 records added, 12 fields written
RHS: Show results nxpdb  in rule 30
```

```
                                 gain                    phase                        correlation
 Name        subject   gain      interesting?   phase    interesting?  correlation   interesting?   nystagmus
••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••
 preflight    s1       0 50                     30 00                  0 85                         0 10
inflight_1             0 35                     24 00                  0 60                         0 10
interesting                      potentially               no                         certainly
••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••
```

Figure 5: Parameter values for a pre- and an inflight experiment are compared. The system suggests which discrepancies should be viewed as interesting.

ORIGINAL PAGE IS
OF POOR QUALITY

Currently, the system detects cases in which the values for attributes obtained inflight deviates significantly from those obtained through preflight trials. In the future, the system will combine this information with an underlying model of the scientific hypothesis being tested, and of the experimental process, in order to explain the deviations that are identified as being interesting.

Protocol Manager

The protocol manager will be the hub of the system. Under normal circumstances it will simply inform the astronaut on which experiment should be performed next and provide any necessary instructions. The real power of this subsystem will come into play in the very likely event that the experiments will not proceed quite as planned. These circumstances include cases when the experiments are taking more or less time than expected, problems are found that take a long time to diagnose, or interesting events are detected. In all of these cases some decision must be made as to the course of action. For example, do we continue to troubleshoot a problem or proceed with degraded data? Given that we have extra time or something interesting has been found what experiment do we perform next? Given that we are running late what experiment do we eliminate, or how do we modify subsequent experiments? The protocol manager will exchange information with the other subsystems, such as the diagnostician or the planner, and provide the mission specialist with alternative courses of action.

Future Development

Short range plans for development include further testing and refining of EOG signal interpretation rules, and extending signal quality monitoring, diagnostics and interesting-case detection to a new set of experiments involving the sled. A prototype of the protocol manager subsystem is also being developed. We do not yet have firm ideas on the planning/re-planning, and experiment suggestion subsystems, but we hope to demonstrate the utility of our work as early as 1990, in the pre- post- flight Baseline Data Collection Facility measurements, as part of Dr. Young's experiments on Spacelab SLS-1. At the earliest opportunity, mission specialists will also become actively involved in helping us design or modify the system.

Conclusions

A set of vestibular experiments planned for future Spacelab missions are providing a test of the applicability of AI technology for increasing the productivity of mission specialists and improving the quality of the data obtained. The functions of a knowledge-based system planned to assist with these experiments will include protocol management, signal quality monitoring, problem diagnosis, detection of interesting cases and planning. The intelligent aspects of these functions are normally considered the province of a principal investigator, hence the system name PI-in-a-Box. The initial prototyping effort has concentrated on signal quality monitoring, problem diagnosis and interesting case detection for a small subset of experiments. A level of system ability sufficient for meaningful demonstrations in these areas has been achieved to date. It appears that the most difficult problem is to extract from the raw instrument data the high level attributes a scientist is accustomed to thinking about. A careful analysis is required in order to distinguish between those attributes that are more easily obtained algorithmically, those that require heuristic knowledge and those that require a deeper underlying model of the scientific hypothesis and of the experimentation process.

Acknowledgements

This project began with seed money from a MIT/NASA University Consortium Agreement in 1987-88, while Dr. Young was on sabbatical leave at Stanford University. The Artificial Intelligence Research Branch at NASA-Ames has provided initial personnel support. The comments that Drs Peter Friedland, William Gevarter and Hamid Berenji of NASA-Ames have kindly given on this paper are greatly appreciated.

References

1. Massoumnia M.A., "Detection of Fast Phase of Nystagmus Using Digital Filtering", M.S. Thesis, Dept of Aeronautics and Astronautics, MIT, May 1983

2. Nexpert Object Version 1.0, Neuron Data Inc., 444 High St., Palo Alto CA. 94301, U.S.A.

3. Young L.R., Oman C.M., Watt, D.G.D., Money, K.E., Lichtenberg, B.K., Kenyon, R.V., Arrott, A.P. "M.I.T./Canadian Vestibular Experiments on the Spacelab-1 Mission: 1. Sensory Adaptation to Weightlessness and Readaptation to One-g: an Overview", Experimental Brain Research 64: 291-298, 1986.

N89 - 15599

ARTIFICIAL INTELLIGENCE APPLICATIONS IN SPACE AND SDI - A SURVEY

By Harvey E. Fiala
SBI Software Engineering
Rockwell International

ABSTRACT

The purpose of this paper is to survey existing and planned Artificial Intelligence (AI) applications to show that AI applications are sufficiently advanced for 32% of all space applications and SDI (Space Defense Initiative) software to be AI-based software [3].

To best define the needs that AI can fill in space and SDI programs, this paper enumerates primary areas of research and lists generic application areas. Current and planned NASA and military space projects in AI will be reviewed. This review will be largely in the selected area of expert systems. Finally, direct applications of AI to SDI will be treated. The conclusion covers the importance of AI to space and SDI applications, and conversely, their importance to AI.

INTRODUCTION

AI covers a broad spectrum of areas of research and accordingly can find many different applications in space and SDI programs. AI includes problem solving, intelligent search strategies, natural language processing, speech recognition, robotics, knowledge and expert systems, computer vision, symbolic processing, logical reasoning, knowledge representation and understanding, and neural network systems. AI applications generally use what might better be termed knowledge rather than data, and frequently use heuristics (which guide trial-and-error searches) rather than algorithms. Some of the above areas of research may not be unique to AI, but they all come under its broad scope.

Using the principles and techniques of AI, generic application areas that have been successfully exploited or are currently under development include analysis, control, debugging, design, diagnosis, repair, instruction, training, interpretation, monitoring, planning, predicting, search techniques and synthesis.

NASA AI PROGRAMS

NASA has stepped up its emphasis on AI research. Its Ames Research Center, Mountain View, California [10] has been given the assignment of taking the lead in AI research. Ames has been directed to assure that its research supports what NASA is currently doing, and also its long-term requirements. NASA has placed a high priority on building "smart" systems for the future US/International space station.

Machine reasoning in uncertain situations, computer learning, and advanced AI architectures are major thrusts in the National Aeronautics and Space Administration's Information Sciences Division at Ames.

The Information Sciences Division at Ames has three sections and concentrates on basic research. The three sections in the division at Ames Research Center are the AI Research Branch, Intelligent Systems Technology Branch, and Systems Autonomy Demonstration Project Office.

At the Johnson Space Center, Houston, NASA concentrates on applying today's AI technology to existing systems.

This paper contains abstracts on expert systems from the publication 'Advanced Military Computing', reference [4], as well as other sources. The different expert systems range in maturity from proof of concept to operational. The author generally does not have information on the current status of the different systems.

SPACE TRANSPORTATION EXPERT SYSTEMS [4]

AIM. The Air Force Space Division in Los Angeles is proceeding with the procurement of an expert system called AIM that will plan the military space transportation systems for the period 1995 through 2010. AIM will be required to design a total "space force" including vehicle use, payload planning, and required support.

STALEX. The Shuttle Trajectory and Launch Window Expert System automates planning of space shuttle missions which carry geosynchronous communications satellites. Developed by the Houston Astronautics Division of McDonnel Douglas Technical Services Co., it performs launch window analysis and payload deployment scheduling. It is supposed to consistently choose the same solution as that of an experienced mission planner and was used for four space shuttle missions before flights were stopped due the Challenger accident in 1987. The expert system reduced planning time by 70% over previous computer tools.

ONEX. An Onboard EXpert System (ONEX) under development for the Johnson Space Center, Houston, will automate many of the navigational monitoring tasks aboard the space shuttle, particularly during routine rendezvous operations.

ESFAS. The Expert System for the Flight Analysis System (ESFAS), created for JSC, Houston, acts as a front end to their conventional software planning program, the Flight Analysis System (FAS). The FAS requires a high level of user expertise, but ESFAS makes its operation simple.

MARS. The Management Analysis Resource Scheduler (MARS), is designed to schedule the resources needed for each space shuttle mission. Resources include manpower, computers, control rooms, simulators, and communications. equipment. It can spot potential problems in planning future missions and was built by Ford Aerospace and Communications Corp., Houston.

PEGASUS. The Prototype Expert Ground Analysis and Scheduler with User Support (PEGASUS), built by the Harris Coproration, is used at the Kennedy Space Center (KSC) to allocate limited amounts of equipment and other items among space shuttle flights.

EMPRESS. The Expert Mission Planning and Replanning Scheduling System (EMPRESS), built by the MITRE Corporation, is used at KSC to determine the time, resources, and tasks required to process payloads for the space shuttle.

VALEX. The VALidate EXpert System (VALEX) is used at JSC to validate software used for shuttle crew training. It discovers if a programmer has entered the wrong wind profile for winds common to the launch area, inaccurate launch weight, the wrong launch pad number or even launch site. Errors include recording the launch site as Florida when it is California.

RENEX. The RENdezvous EXpert System (RENEX) to automate rendezvous of space vehicles is being developed by JSC's Mission Planing and Analysis Division. It will graphically display what a system or program does and what data it uses and creates, and will be used for ground pre-mission planning, ground real-time planning and monitoring, and onboard planning and monitoring.

NAVEX. The NAVigation EXpert system is used to assist in the high-speed ground navigation of a shuttle flight during the ascent and entry portions of a flight. It involves monitoring and processing data from radar stations which are tracking the shuttle. It was developed by Inference Corp.

EXEPS. The EXpert Electrical Power System (EXEPS) is under development by NASA to aid in scheduling spacecraft electrical power loads. The Mission Planning and Analysis Division is responsible for characterizing the expected performance of the electrical power system before each flight, based on that flight's schedule of trajectory events and crew activities. This job currently takes up to two weeks to do manually. The expert system will function as an intelligent advisor and focuses on the man-machine interface and planning technology.

MISSION CONTROL SOFTWARE MONITOR. NASA uses this expert system to monitor the Mission Control Center software status. This job is currently done by the printer controller. The expert system will recognize which of the print status and error messages are important, what they mean, to whom they are important, and will report the appropriate status information.

HEAT. The Heuristic Error Analyzer for Telemetry (HEAT) looks at a compressed image of the telemetry status, as output by the network data driver at mission control. It automatically distinguishes routine errors from those requiring human attention.

SATELLITE CONTROL AND TRACKING EXPERT SYSTEMS [4]

RIACS is taking part in joint projects to apply expert systems to aerospace problems. Four of these projects are: automatic grid generation being done for the Applied Computational Aerodynamics Branch, mission planning for the Infrared Telescope, automatic programming of space station systems for the Space Technology Branch, and evaluation of aircraft configurations for the Aeronautical Systems Branch.

AUTONOMOUS SATELLITE CONTROL. An expert system called Autonomous Satellite Control is being developed by the Space Technology Center, at Kirtland Air Force Base. The expert system is intended to allow a satellite to operate on its own for up to 30 days. The expert system, carried onboard a communications, navigation or reconnaissance satellite, will allow fine tuning of orbital parameters to maintain orbit, recharge batteries and perform other routine tasks that now require ground controllers.

TDRS CONTROL SYSTEM. This expert system is being developed by Contel-Spacecom and General Research and will aid in maintaining the orbital position of the Tracking and Data Relay Satellite used by the space shuttle, and should also be applicable to other satellites. It will be able to guide a satellite controller through a maneuver required to maintain the proper East-West orientation of the satellite. It monitors errors, manages thruster temperatures, monitors gyro performance, informs the controller of what is happening, and recommends steps to follow.

ESCAP. The Element Set Correction Assistant Prototype (ESCAP) expert system automatically updates database information after reading tracking sensors. It shows where orbiting objects actually are, compared to computer predictions of those orbits. This expert system was demonstrated to the Air Force Space Command by Ford Aerospace and Communications.

SCARES. The Spacecraft Control Anomaly Resolution Expert System (SCARES) expert system uses real-time monitoring to detect anomalies at an early stage. Then, using diagnostics, it localizes a fault to a specific unit or assembly, and a hypothesis generation and testing function analyzes faults not easily resolved by the diagnostic process and suggests recovery actions. It handles only anomalies in a satellite's attitude control. Planned extensions would monitor and diagnose the thermal, power, and payload systems.

MOPA. The Mission Operations Planning Assistant (MOPA) expert system will aid the Mission Planning Group at Goddard Space Flight Center to do mission planning for the Upper Atmospheric Research Satellite, to be launched by NASA in the early 1990s. It will help coordinate the daily activities for 10 instruments over an 18 month mission life.

ESSOC. The Expert System for Satellite Orbit Control (ESSOC) gives real-time aid to ground control operations. It provides continuous analysis of telemetry data throughout the stationkeeping operation by means of two-way real-time communication. It provides autonomous processing for satellite maneuvering operations. The system handles phase independent and phase-specific functions. When ESSOC makes a recommendation the user can choose to override it or ask for additional information before taking action.

SPACE STATION EXPERT SYSTEMS [4]

AMPASES. The Automatic Mission Planning and Scheduling Expert System (AMPASES) was developed to provide scheduling activities for NASA's space station. It can set up a 24-hour schedule in less than 30 minutes and a one-week schedule in an hour or two. Finding the "best" schedule may not be possible with the limited computing resources available, due to the large number of variables involved in managing space station missions. The expert system develops a high quality, reasonably efficient schedule based on the constraints and data provided. It can deal with seven crew members, 14 resources of various types, and three types of interactions among missions.

ECESIS, AESOP. The Environmental Control Expert System in Space (ECESIS) provides control of life-support systems on a space station. The Autonomous Electrical Subsystems OPerational (AESOP) expert system monitors, diagnoses, and controls the space station electrical system. Both were developed by Boeing Aerospace Co.

SSES. The Space Station Expert System (SSES) was developed to provide on-board advice to space station operators when reconfiguring for unexpected, hazardous or resource-threatening events. It works in conjunction with a simulation of the space station operator's command module, and was developed by Lockheed Engineering and Management Services Co.

SPACE STATION HOUSEKEEPING. NASA has developed a space station expert system for housekeeping and maintenance on a 90-day mission. It creates a recurrent priority-ranked housekeeping schedule for long term space missions.

FREE-FLYER CONTROL. Engineers at Mitsubishi have built an experimental expert system for tracking and controlling any type of satellite, for checking out equipment and other systems, and for operating and controlling free-flyers.

ADEPT. The Automatic Detection of Electric Power Troubles (ADEPT) expert system integrates knowledge from three different suppliers to offer an advanced fault detection system, and is designed for two modes of operation: real time fault isolation and simulated modeling [11]. The system has been tested with a simulated space station power module.

ROCKWELL INTERNATIONAL AI SPACE PROJECTS

EXCABL. Rockwell International has developed an expert system for automatically generating the payload bay cabling design for each mission of the space shuttle [8] [9]. This expert system, in production since 1986, requires less than one person-hour to complete a cable design, compared to 6 person-days previously.

EXMATCH. The EXpert Drawing MATCHing System (EXMATCH) is used on the shuttle program to automate the payload bay cabling installation Technical Order (TO) generation task [8]. Closely integrated with the EXCABL system, the cabling solution provided by EXCABL is automatically input to EXMATCH, and a master listing of all required payload cabling installation TO's is generated.

EXTOL. The Technical Order Listing EXpert System (EXTOL) is planned for development at Rockwell in the near future [8]. EXTOL will produce the initial Mission Equipment Cargo Support Launch Site Installation drawing. It will also use heuristics and data from previous missions to assist the user by producing a list of the best matches for mission TOs in the mission-unique category.

SAFE. Under development at Rockwell International for use on the space shuttle air revitalization system is the Safing and Failure Detection Expert (SAFE) [5], [15], [16]. It is designed to be crew-operated and uses a touch screen interface.

RB-ARD. The Rule-Based Abort Region Determinator (RB-ARD) for real-time monitoring and control, as a support system for ground personnel [17].

EXTRAJ. The Ascent Throttle Bucket Designer (EXTRAJ) is an expert system designed to optimize the shuttle throttle profile during ascent [14]. The throttle is set at full at liftoff (to minimize gravity losses), is reduced to limit dynamic loading, then returned to full as the atmospheric density falls off with altitude. The resulting thrust profile is called a "thrust bucket", and its shape influences fuel consumption.

DDES. The DDES is an expert system under development for the management of data displays in ground-based shuttle mission support [7].

ICA. The Intelligent Communicating Agents (ICA) expert system, currently under development, will help communication between distributed subsystems, and is intended for use by NASA to help provide autonomy for a manned mars mission experiment.

ETAR (ROBOTICS). The preliminary conceptual design of a new teleoperator robot manipulator system for Space Station maintenance missions has been completed at the Automation and Robotics Laboratory at Rockwell International, Downey, California [1].

The system consists of a unique pair of arms that is part of a master-slave, force-reflecting servomanipulator. This design allows greater dexterity and greater volume coverage than that available in current designs and concepts.

SDI AI APPLICATIONS

Rockwell International (RI), Downey, California, has SBI (Space Based Interceptor) contracts with the Air Force to produce a light weight prototype Kinetic Kill Vehicle (KKV), as the major defensive weapon in this nation's SDI program, and define the System Concepts and Integrated Test (SCIT) program, which includes defining the software for an operational SBI program. Previously, RI had conducted its own Independent Research and Development program to determine the extent to which AI is applicable to a space or ground-based interceptor program [2].

On the SBI program, RI is presently using an internally developed expert system called ADAES (Advanced Data Assessment Expert System) to help with the tremendous amount of data that is generated on the program. ADAES automates on-line data reduction, data assessment, and trend analysis functions for complex simulations. It allows intelligent data base management of initial conditions, initialization and control of simulation, real-time data monitoring, post-run performance analysis, multiple run comparison analysis, and provides operations and test teams with clear visibility of simulation performance.

SDI related software can be divided into two distinct classes: OPERATIONAL and NON-OPERATIONAL (SUPPORT) software [2].

Operational software is defined to be critical software which will be used before, during, and after an SDI battle or in support of space applications. It is comprised of actual flight software, some ground-based, some airborne, and some space-based. Support software is defined to include all other software (probably all ground-based) that is associated with supporting the life cycle of the operational software.

Candidate areas for applying AI to operational SDI software include battle station performance, weapon platform performance, flight control, constellation architecture design, end game, battle management, guidance, navigation & control, data processing, communications, and electrical power management. Other applications include off-line system development and management, Surveillance, Acquisition, Tracking, and Kill Assessment (SATKA), weapons fault analysis, and parameter tuning, and in battle management, command and control to make assumptions about "who's doing what to whom and why."

Support software that can be AI-based include automated code generators for the development phase, verification software for the verification phase, maintenance software for the maintenance phase, and testing and monitoring software for the operational phase. A detailed discussion of SDI software producibility including a definition of the term 'software producibility' is contained in reference [3]. Personnel training software under the title of CBT (Computer Based Training) is also an example of support software.

Other generic SDI support software applications include expert and knowledge systems for boosters, SDI resources, laser weapons, phenomenology, sensors, threats, weapons, defense strategy, G2 (military intelligence), and suppliers. Expert systems normally capture their knowledge from experts, while knowledge systems generally get their knowledge from documented sources [6]. Applications include expert systems that perform the functions of communications, control, instruction, training, monitoring, planning, predicting, and threat-alert. Knowledge based workstations for threat analysis, threat alert, and defense strategy are also important SDI applications. Other applications include robots that perform remote maintenance, remote monitoring, and remote threat-alert. The importance of Artificial Intelligence to space defense was understood before SDI was established [13].

It is estimated that an early SDI system might have about 3.0 million lines of code (MLOC) of operational software and about 5.0 MLOC of support software [3]. Based upon the general capabilities of Knowledge-Based AI systems, it is estimated that about 45% of the lines of code for the support software and only about 10% of the lines of code for the operational software can be AI based software. This results in about 2.25 MLOC for the support AI software and about 0.3 MLOC for the operational AI software. The total SDI AI software is then 2.55 MLOC or about 32% of the total of 8 MLOC [3].

The beginning of the development of an operational SDI system, if funded, is approximately three years away and it will be about ten years (at the earliest) before a fully operational system, whether space or ground based, can be completed. This time interval clearly allows for advances in the speed of AI processors, such as LISP machines, supercomputers, parallel processors, optical computing etc., and in the maturity of expert systems and knowledge based tools for automated code generation and support of the full software development life cycle.

CONCLUSIONS

AI can and will be very important to space and SDI, and space and SDI can be very important to AI. There have been premature conclusions regarding the feasibility of software including AI software for the SDI program -- for example, David L. Parnas' essays of June 1985 [12]. It is the author's belief that several of the reasons given in Parnas' essays are not correct [3]. Furthermore, almost three years of progress have occurred since then, and a much more realistic and mature perspective is now available. For an "early-deploy" SDI system, the complexity of the operational software will be significantly less, thus allowing a normal maturation process by going in phases from an early-deploy system to a full-scale SDI system.

It is possible that the SDI program could be significantly watered down or even cancelled for entirely invalid political reasons. If this should occur, it would also be a serious blow to AI technology. The SDI program offers so many natural opportunities for the legitimate application of many different expert and knowledge systems, and accordingly, it can give AI the boost that it needs to become a mature and accepted technology. AI maturity is a significant 'spinoff' of a space station or an SDI program.

Considering both the operational and support software, a study by the author indicates that as much as 32% can be AI based. This same percentage is conservatively applied to space software. This is an opportunity for AI to mature that this nation should not let slip by. The short space allowed for this paper allows only touching upon the highlights of applying AI to space and SDI. It would require much more than this paper to do justice to this cause.

Mr. Fiala has a Master's Degree in Electrical Engineering from the California Institute of Technology, a Certificate in AI from UCLA, is a Registered Professional Engineer in the State of California, and has been a software engineer for eighteen years at Rockwell International, Downey, California.

Gratitude is expressed to Greg Ekstrom, Keith Morris, Burton Smith, and Jonathan Post for their helpful comments after having read the various drafts of this paper.

REFERENCES

[1] Clarke, M. M., C. J. Divona, and W. M. Thompson, "Manipulator Arm Design For The Extravehicular Teleoperator Assist Robot (ETAR): Applications on the Space Station", presented at the Space Operations-Automation and Robotics (SOAR) Workshop, Houston, Texas, 5-7 August 1987, Rockwell International paper number SSS 87-0095.

[2] Ekstrom, G. J., "Preliminary KEW Software Analysis Report", Rockwell Internal Letter from G. J. Ekstrom to I. Himmelberg, March 10, 1985.

[3] Fiala, H. E., "SDI Software Producibility", Rockwell International Second Annual Software Engineering Symposium (SES II), April 19, 1988.

[4] Keller, J., A. K. Marsh, and J. Vedda, Applied Aerospace & Defense Expert Systems (abstracted from Advanced Military Computing), 1987, section 3, pages 109-149.

[5] Koch, D., Morris, K., Giffin, C., and Reid, G. T., "Avionic Sensor-Based Safing System Technology", Presented to Tri-Service Software Systems Safety Working Group (July 7, 1986) and IEEE Computer Assurance (COMPASS) Conference (July 10, 1986), Washington D. C.

[6] Mishkoff, H. C., Understanding Artificial Intelligence, second printing, 1985

[7] Morris, K. E., and Giffin C., "Expert Systems Applications for Today's Space Problems", presented at the 1988 International Computers in Engineering Conference, Real-World Applications of Expert Systems and Artificial Intelligence, August 1-4, 1988, San Francisco, California.

[8] Morris, K. E., "Expert Systems Applications for Space Shuttle Payload Integration Automation", presented at the Second Annual Space Operations Automation and Robotics Workshop (SOAR), Wright State University, Dayton, Ohio, July 20-23, 1988.

[9] Morris, K. E., Nixon, G.A., and Rejai, B., "EXCABL--Orbiter Payload Bay Cabling Expert System, A Case Study". Proceedings of IEEE Westex-87 Expert Systems Conference, Anaheim, California, IEEE, (1987).

[10] NASA Ames consolidates AI research, Advanced Military Computing, June 22, 1987, pages 5, 6.

[11] NASA, Second Annual Space Operations Automation and Robotics Workshop (SOAR 88), Abstracts for Technical Sessions, Wright State University, Dayton, Ohio, July 20-23, 1988.

[12] Parnas, D. L., "Software Aspects of Strategic Defense Systems", American Scientist, September-October 1985, Pages 432-440.

[13] Post, J. V. "Cybernetic War,", OMNI Magazine, May 1979.

[14] Quan, A., and Smith, R., "An Expert System for the Design of Space Shuttle Ascent Trajectory Throttle Buckets". Proceedings of IEEE Westex--87 Expert Systems Conference, Anaheim, California, IEEE, (1987).

[15] Reid, G. T., and Nixon, G. A., Final Report FY 1985, Automated Malfunction Procedures System, Rockwell International, STS 85-0478 (1985).

[16] Reid, G. T. et al, FY 1986 Final Report, SAFE-Human Interface Enhancement, Rockwell International, STS 86-0325 (1986).

[17] Smith, R., and Marinuzzi, J. Final Reports for IR&D Projects 87267. Rockwell International, IL-282-900-GLN-87-001(1987).

N89 - 15600

CONCEPTS for AUTONOMOUS FLIGHT CONTROL for a BALLOON on MARS

Thomas F. Heinsheimer, TITAN Systems
Robyn C. Friend, TITAN Systems
Gardena, California

Neil G. Siegel
Redondo Beach, California

ABSTRACT

Balloons operating as airborne rovers have been suggested as ideal candidates for early exploration of the Martian surface. An international study team composed of scientists from the U.S.S.R., France, and the U.S.A. is planning the launching in 1994 of a balloon system to fly on Mars. The current likely design is a dual thermal/gas balloon that consists of a gas balloon suspended above a solar-heated thermal balloon. At night, the thermal balloon provides no lift, and the balloon system drifts just above the Martian surface; the lift of the gas balloon is just sufficient to prevent the science payload from hitting the ground. During the day, the balloon system flies at an altitude of 4 to 5 kilometers, rising due to the added lift provided by the thermal balloon.

Data from the NASA/Ames Global Circulation Model (GCM) indicate that over the course of a single Martian day, there may be winds in several directions, and in fact it can be expected that there will be winds simultaneously in different directions at different altitudes. Therefore, *a balloon system capable of controlling its own altitude, via an autonomous flight control system, can take advantage of these different winds to control its direction*, thereby greatly increasing both its mission utility and its longevity.

Once autonomous balloon control has been demonstrated in the 1994 mission, ever more complex balloon missions could follow, at each Mars launch opportunity, paving the way for a Mars sample return mission near the end of the century, and later, for manned exploration of Mars.

This paper describes the trade-offs and issues involved in the balloon-autonomy concept, investigates the benefits to the mission of various levels of autonomy, and describes concepts under consideration and the current program plan. The authors conclude that a goal-driven autonomous control system for the Mars balloon can be constructed, and that the use of such a control system would increase the effectiveness of the scientific return, the safety of the balloon system, and the duration of its mission on Mars.

1. INTRODUCTION

A balloon provides an effective platform for Mars surface exploration in that it is potentially long-lasting, and can travel many miles over its lifetime. The balloon mission complements the rover, in that that latter carries a much larger science payload over a shorter trajectory, whereas the balloon can explore a much larger area with a smaller payload. Together they will collect science data that cannot be observed by Mars orbiters.

The international study team (the U.S.S.R., France, and the U.S.A.) is presently analyzing four balloon designs, including a dual thermal/gas balloon, a single superpressurized balloon, a classic (i.e., ambient pressure) gas balloon, and a kite balloon (Figure 1). The concepts of autonomy discussed here are applicable to all balloon designs; the most likely candidate from among these four designs is the dual thermal/gas concept, which will be assumed for the purposes of this paper.

While the balloon system is flying, using combined gas and thermal lift, the horizontal component of its movement is dependent on wind direction and velocity. Data from the NASA/Ames Global Circulation Model (GCM) indicate that over the course of a single Martian day, there may be winds in several directions, and in fact it can be expected that there will be winds simultaneously in different directions at different altitudes [PolJ87]. Therefore, *a balloon system capable of controlling its own altitude, via an autonomous flight control system, can take advantage of these different winds to control its direction*, thereby greatly increasing both its mission utility and its longevity. Whereas the simple daily ascent to ceiling altitude of the proposed balloon design will provide an interesting science return, a navigable balloon system could be programmed to achieve a broad range of additional missions, such as documentation of candidate landing sites, collection of documented samples and their return to a common area, and visits to areas of special interest as a result of other observations (such as those from the Mars Observer, etc.). Devising an autonomous flight control system for the dual thermal/gas Mars Balloon would greatly enhance the balloon system's scientific return.

2. AUTONOMOUS CONTROL of the DUAL THERMAL/GAS BALLOON

The dual thermal/gas balloon design consists of two balloons, one gas-filled, the other a solar-heated thermal balloon. The closed gas balloon is suspended above the solar-heated thermal balloon (Figure 2). At night, the thermal balloon provides no lift, and the balloon system drifts over the Martian surface; the lift of the gas balloon is intended to be just sufficient to prevent the science payload from hitting the ground. The ground-sampling portions of the science payload instruments can be operational during this time (Figure 3) [FriRC87].

At sunrise, the thermal balloon is heated by sunlight, complementing the lift of the hydrogen-filled balloon, and causing the entire vehicle to rise to its float altitude, where it cruises all day. In the afternoon, as the sun begins to set, the thermal balloon cools, thereby losing its lift; the balloon vehicle drifts down to the surface, where it again moves along the surface on its ground payload. The gas balloon still prevents the science payload from being damaged by hitting the ground. (Figure 4).

Some degree of flight control is needed just to assure a safe flight. Some situations where control would be beneficial include the following:

- o Morning liftoff prior to excessive soloar heating, to avoid buffeting by thermals, etc., which results from taking off when the atmosphere is hot. Such buffeting could pose a danger to the integrity of the thin balloon envelope. Note that there is a high level of uncertainty concerning actual Martian wind and weather conditions.

- Stable ascent and float at ceiling altitude.
- Controlled daytime altitude.
- Initiation of descent after thermals have cooled, but before the air is so cool that the balloon system crashes.
- Controlled landing.
- Nighttime traverse.
- Compensation for deterioration in thermal balloon envelope (i.e., resulting from buffeting over time).
- Compensation for gas leakage from gas balloon.

The descent and ballast issues are of particular interest. As the sun sets, the balloon starts to descend from its cruising altitude of 4 to 5 kilometers. With time the descent rate tends to increase due to two effects: reduced solar elevation provides less solar heat; and forced convection cooling of the descending solar-heated balloon reduces its lift. If unchecked, the result is an approximately 5 meter/second descent rate culminating in the crash of the balloon onto the surface.

To avoid this disaster a design is needed that starts the descent early enough in the afternoon to assure sufficient solar input all the way down, and which prevents a "runaway descent" by keeping descent velocity at less than 1 meter/second. This could be accomplished either by a clever design of a "dumb" system that has no active control, or by various levels of active control. The tradeoffs of these alternatives are under discussion.

Another control issue involves the use of ballast to compensate for gas losses in the gas balloon, and for losses in the efficiency of the thermal balloon due to degradation of the envelope film. The decision of when to deploy ballast, and how much is to be deployed, must be made efficiently -- either in a balloon-borne controller or from Earth. If on the balloon, it requires appropriate devices; if on Earth, it requires extensive two-way communications.

Aside from flight safety and longevity issues, the performance of the science payload may be enhanced by the actions of the autonomous controller, for example:

- Modifying the flight trajectory.
- Modifying dwell-time at a landing site.
- Managing the limited budget of power, data storage/processing, communications times, system lifetime.

Such actions may increase in sighting opportunities, increase the accuracy or interest of the measurements, and prolong the life of the science payload.

The use of the autonomous controller to manage the flight trajectory is particularly interesting. At morning liftoff the balloon will perform an ascent to cruise altitude that would allow the measurement of the wind profile. This profile will have a validity interval of several hours, during which time the trajectory can be influenced by the proper choice of altitude profile. An interesting issue is the degree of autonomous control that should be resident in the balloon, and how much in a large earth-based computer having a wind model of Mars, how much reliance on in situ measurements and how much on model-based predictions. The value of controlling the trajectory is clearly to improve the science return by going towards interesting areas, and to protect the balloon by avoiding treacherous terrain that could either be hit in the daytime, or cause system damage during guide-rope traverse at night.

3. AUTONOMOUS CONTROL: CONCEPTS and ISSUES

The issue of autonomous control must be viewed in the context of a series of ever-more complex balloon flights, occurring roughly every two years, for the long-term exploration of Mars. At each step, increasing levels of autonomous control may be appropriate in order to achieve mission goals.

Due to the low density of the atmosphere of Mars (approximately 10 grams/meter^3 at the surface), the mass of the balloon system must be kept as small as possible. One must carefully assess the impact on the over-all scientific mission of the balloon system of any additional sub-system, such as the autonomous controller. The current design achieves a science payload of 20 kilograms out of a total system mass of 70 kilograms; to achieve even this level of science-to-mass efficiency, the balloon "skin" must be only 3.5 microns thick [HeiTF88].

In the previous sections, we have briefly described some of the mission benefits that could be realized through the use of an autonomous control system. The principle "cost" or "down-side" is the additional mass that must be added to the system, much of that will have to take the form of a displacement of science payload. A clever design will, however, use sensors that have a science return, and a processing system that can be shared by the science payload.

The components of the autonomous control system include the following:

- Computer processor and storage devices
- Additional power ration
- Sensors
- Controllers and actuators
- Inter-connections and interfaces

Typical of the information that an autonomous controller may need are the following:

- Solar elevation angle
- Pressure and radar altitude
- Atmospheric turbulence
- Surface roughness
- Vertical profile of horizontal wind
- Balloon system status
- Balloon geographic location

This data can be obtained by a combination of on-board sensing and external input. The impact of on-board sensing on system weight may be severe unless highly-sophisticated sensing/processing is done to minimize weight and power.

The implementation of control is typically done by two types of actions: (1) irreversible jettisoning of ballast (perhaps in the form of 200-gram science modules that fall to the surface as long-term meteorological stations); and (2) reversible modifications of the thermal efficiency of the thermal balloon by either venting internal hot air, or changing the thermal characteristics of the side facing the sun.

The management of the ballast drop is relatively easy, but the thermal control of the hot-air balloon may require careful design. The 3.5 micron film cannot stand much manipulation. One concept under discussion uses a thermal balloon having different films (clear, aluminized, or gold-plated) at various locations around the balloon, and a control system that rotates the balloon with respect to the sun to achieve the desired thermal input. Such a system of controlling ascent/descent velocities is being designed.

4.0 LEVELS of AUTONOMY

The dual thermal/gas balloon can be used for a variety of missions having an increasingly complex level of engineering sophistication and scientific return.

- The simplest mission (i.e., no autonomous controller) makes diurnal ascent/descent, carrying sensors such as a vibration sensor and vibration spectrum analyzer in its ground payload to monitor ground mechanical properties during nocturnal ground traverse, an array of atmospheric sensors measuring temperature, turbulence, dust size and concentration, and additional sensing [BlaJ86].

- A more complex balloon might have a valve in the thermal envelope to allow controllable altitude profiles during the day. After an initial ascent in the morning (as indicated in Figure 5) to determine current local wind patterns, the controller then uses the valve in the thermal bag to select an altitude for the day's flight. This could be extended to incorporate numerous diurnal ascent/descent profiles as well as low altitude (terrain-following) missions with the aid of a small radar altimeter to permit high-quality ground imaging using a CCD camera.

- A still more elaborate concept uses an intelligent balloon with an on-board array of sensors and computers that allow it to optimize the science return, by intelligently controlling its vertical and horizontal profile. This could include the dropping of portions of the ground payload mass in the form of small ground monitoring stations, or the collection of ground samples from desirable locations for balloon transport to an area having easy access by a subsequent rover. The "elephant graveyard" mission, in which several balloons are sent on widely separated sample collection missions bringing samples to a central site for later retrieval, is an example of the potential use of such sophisticated balloon systems.

A minimal Mars balloon autonomous control system would consist of a computer processor and storage devices (ROM for programs, RAM for sensor readings and computational products), wind direction and velocity sensors, a controller and actuator to open and close a valve of the thermal balloon envelope, and inter-connections and interfaces between the computer processor, the sensors, the power source, and the controller/actuator.

Without any significant increase in weight, inter-connections could be added between some portions of the science payload and the autonomous controller's computer processor. With the corresponding additions to the controller's software, payload effectiveness could be increased through the mechanisms described in Section 3.1.

Investigations are underway to determine the cost-benefit trade-offs of additional sensors.

5.0 SUMMARY and PROGRAM PLAN

This paper described the benefit, trade-offs, and issues involved in the balloon-autonomy concept, investigated the mission pay-offs of various levels of autonomy, and described concepts under consideration. The authors conclude that a goal-driven autonomous control system for the Mars balloon can be constructed, and that the use of such a control system would increase the effectiveness of the balloon system, both by providing an improved science return and by increasing the level of safety.

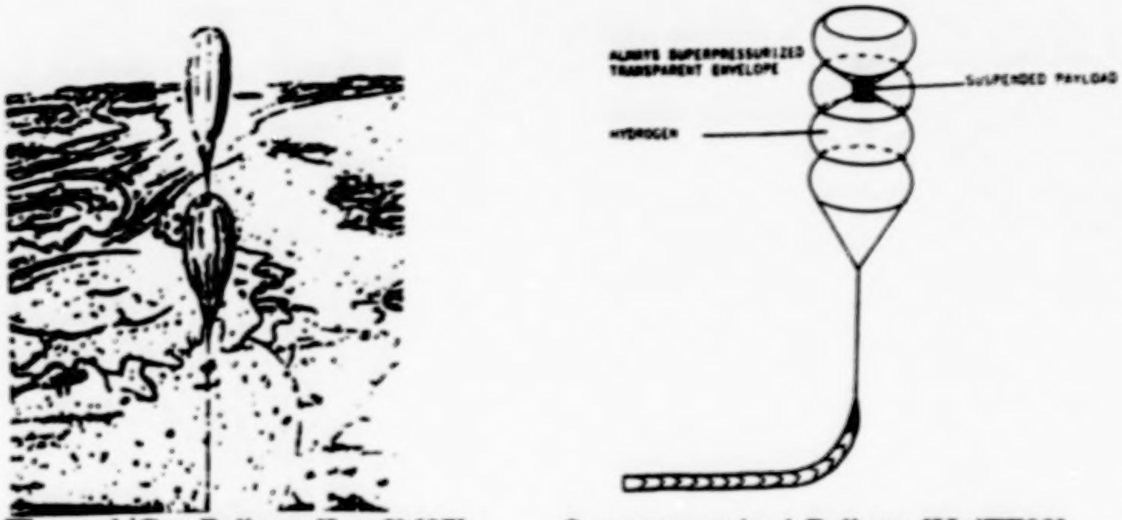
In-progress and near-term activities include:

Construction of trial balloons out of thin film. The French Space Agency (CNES) has successfully flown a 1/10-volume scale-model of the Mars thermal balloon, proving that it can be built, deployed, and inflated (at 30 kilometers above the Earth), and that thermal performance is as expected. Smaller scale models have been provided by CNES to JPL for tests in a ground environment.

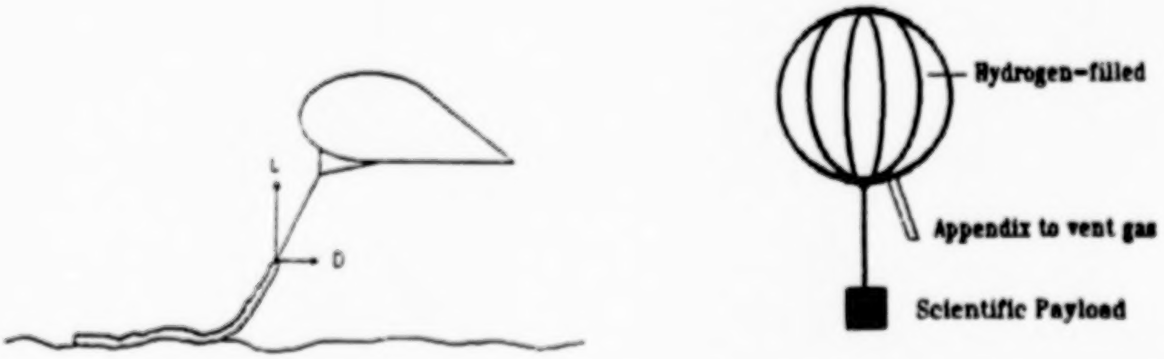
Guide-rope experiments. JPL and Cal Tech are currently working on a second-generation "snake"-type guide-rope, to be tested in the California desert. Sensors and data recorders will be attached to record speed and surface roughness. There has been some discussion of the possibility of using the flexibility of the snake as a means for determining the size of boulders on Mars, and thus determining suitability of prospective sites for use as prospective landing platforms for other vehicles and missions.

Balloon-flight dynamics experiments. TITAN Systems and the Soviet Space Research Institute (IKI) have been conducting balloon-flight dynamics experiments in the Soviet Union, under the sponsorship of The Planetary Society, a California-based organization that promotes space exploration.

Planning of autonomous design concepts. TITAN and others are proposing and analyzing various AI and knowledge-based techniques for providing the autonomous control function.



Dual Thermal/Gas Balloon [LenJM87] Superpressurized Balloon [HeiTF88]



Kite Balloon [BurJD88] Classic Single Gas Balloon [HeiTF88]

Figure 1. The Four Candidate Balloon Designs.

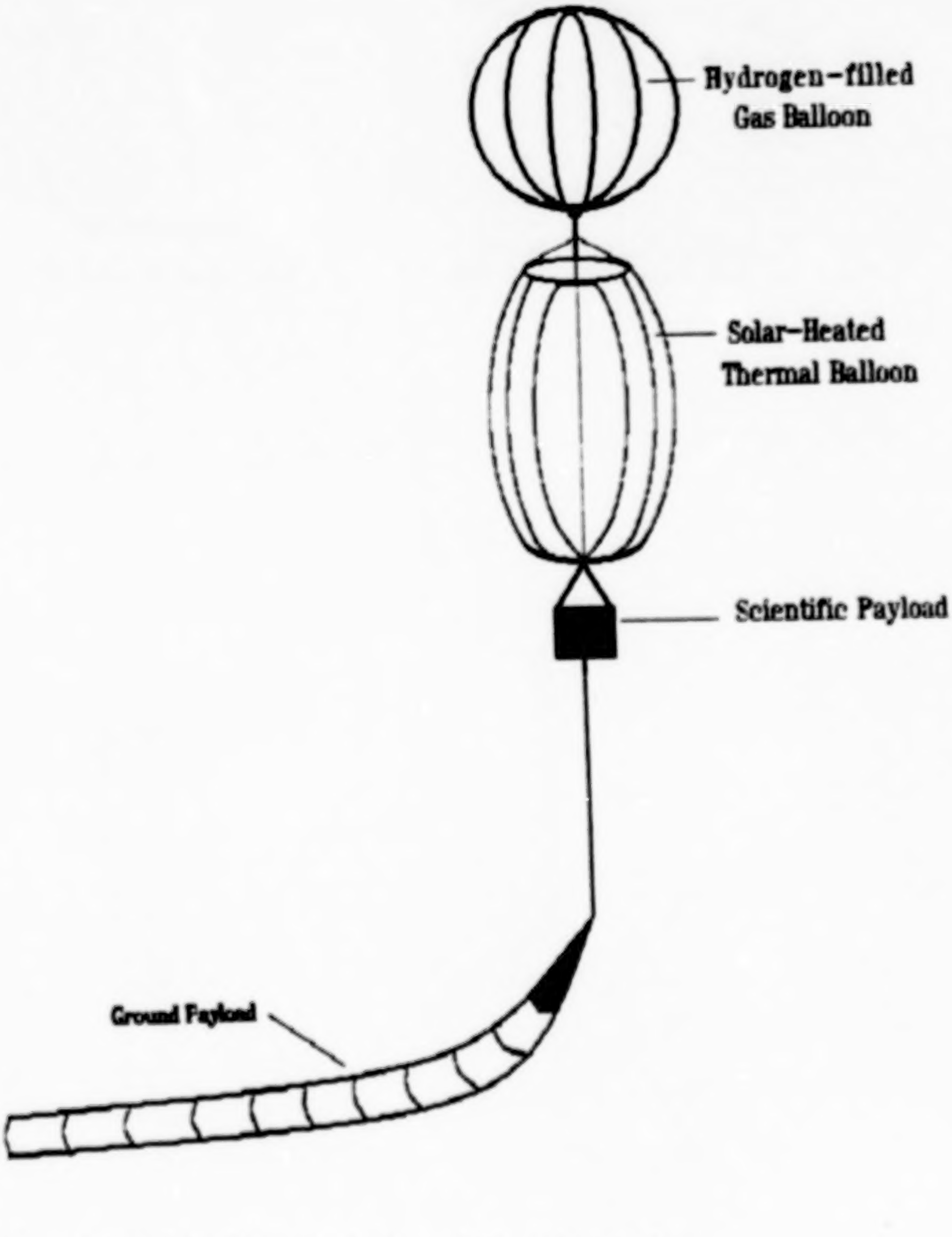


Figure 2. Detail of the Dual Thermal/Gas Mars Balloon.

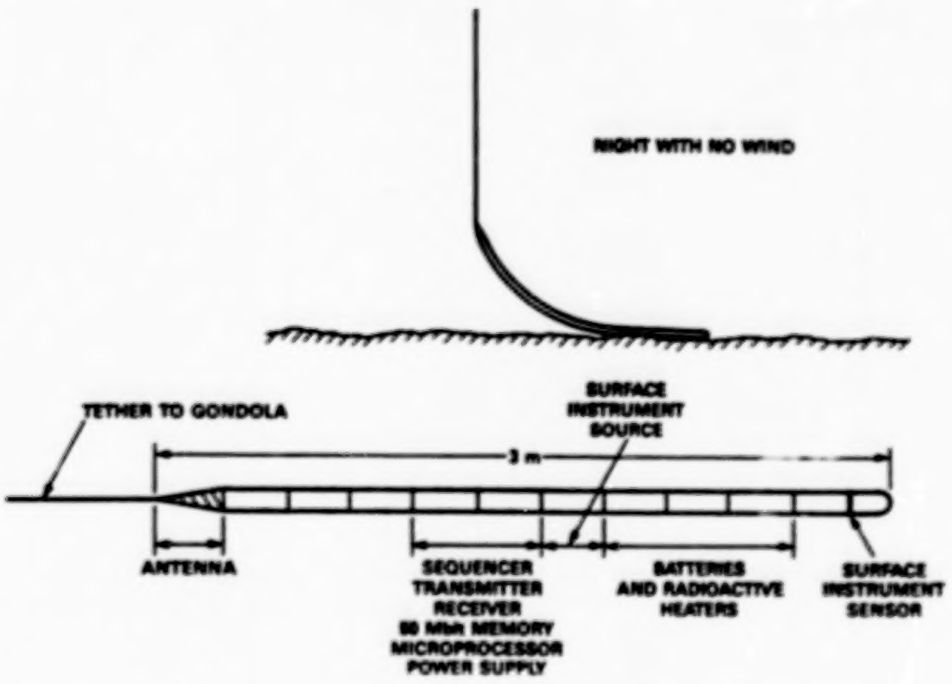


Figure 3. Detail of Ground Payload [AHTR87].

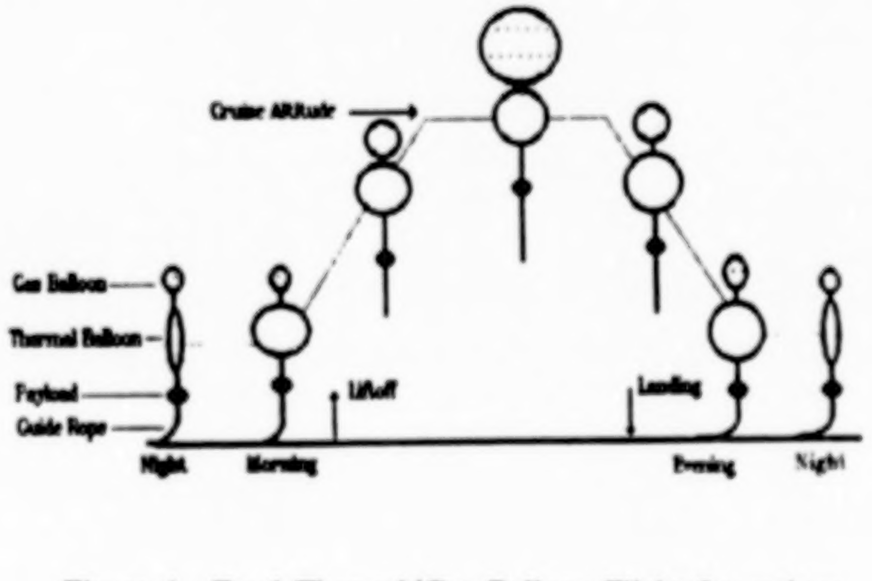


Figure 4. Dual Thermal/Gas Balloon Flight Operations.

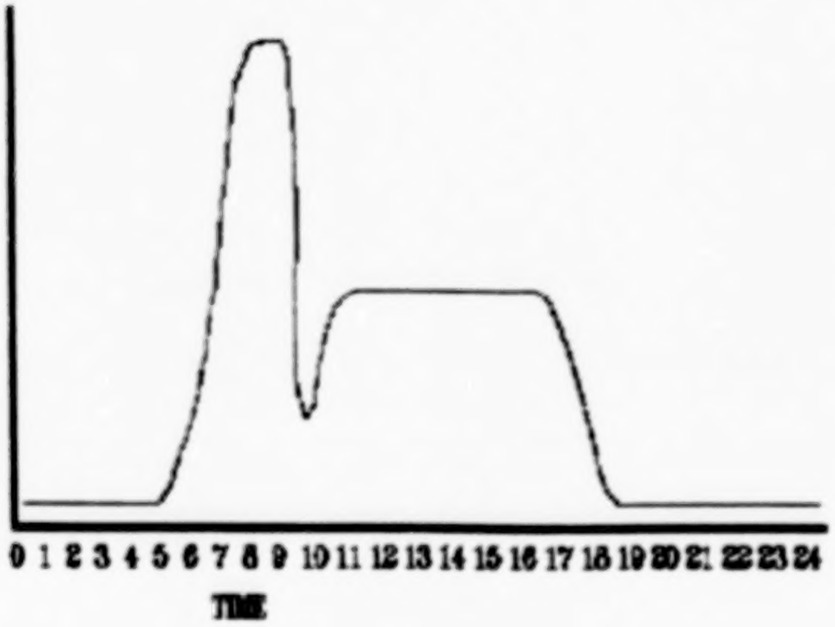


Figure 5. Candidate Single-Day Operating Profile.

The authors wish to aknowledge gratefully the support and encouragement of Jim Burke of the Jet Propulsion Laboratory, and Lou Friedman of The Planetary Society.

REFERENCES

AHTR87 Ad Hoc Team Report, Moscow, October 1987.

BlaJ86 Blamont, Jacques. "Balloons for Mars Missions." Paper IAF-86-324 presented at the 37th Congress of the International Astronautical Federation, Innsbruck, Austria, October 1986.

BurJD88 Burke, James D. "Mars Balloon Status". JPL, 24 February 1988.

FriRC87 Friend, R. C. and Heinsheimer, T. F. "Ballooning on Mars, Report Number 1". TITAN Systems, June 1987.

HeiTF88 Heinsheimer, T. F., Friend, R. C., et al. "Exploration of Mars by Balloon: Four Concepts". COSPAR 1988, Espoo, Finland.

LenJM87 Lenorovitz, Jeffrey M. "French Offer Balloon Platform for Use on Soviet Mars Mission". Aviation Week and Space Technology, 3 August 1987. Drawing by Olivier de Goursac.

PolJ87 Pollack, Jim and Haberle, Bob. "Mars GCM Simulations: Implications for Balloons". Presented in Paris, France, 8 October 1987.

N89-15601

A Very Large Area Network (VLAN) Knowledge-base applied to Space Communication Problems

Carol S. Zander

Department of Computer Science
Colorado State University
Ft. Collins, CO 80523

ABSTRACT

This paper first describes a hierarchical model for very large area networks (VLAN). Space communication problems whose solutions could profit by the model are discussed and then an enhanced version of this model incorporating the knowledge needed for the missile detection-destruction problem is presented.

A satellite network or VLAN is a network which includes at least one satellite. Due to the complexity, a compromise between fully centralized and fully distributed network management has been adopted. Network nodes are assigned to a physically localized group, called a "partition." Partitions consist of groups of "cell" nodes with one cell node acting as the organizer or master, called the "Group Master" (GM). Coordinating the group masters is a "Partition Master" (PM). Knowledge is also distributed hierarchically existing in at least two nodes. Each satellite node has a back-up earth node. Knowledge must be distributed in such a way so as to minimize information loss when a node fails. Thus the model is hierarchical both physically and informationally.

1. Introduction

Distributed problem solving networks provide an interesting and powerful base for solving those problems for which a single problem solver or single machine seems inappropriate. Problems requiring world-wide support, involving network nodes in space fall into that category. While local area and wide area networks currently solve many distributed problems, satellites provide expanded capabilities unavailable with local and wide area networks. In the past, satellites have mostly been simply a reflector of signals, computationally passive in computing networks. However, with the rapid advancement of technology, it is conceivable that in the near future satellites will have on-board computing power. Satellites may contain a single computer or a local area network.

A network containing one or more satellites is classified as a Very Large Area Network (VLAN). Problems that will be solved using the VLAN model are extremely complex in part due to the large spatial distribution of nodes. To manage the network and control the information distributed throughout the system, artificial intelligence techniques are needed. A knowledge-based system manager handles the duties of resource management and network communication. This paper introduces the VLAN model and applications, but details of the manager will be presented in later work.

In the next section, the VLAN model is presented with its organization: the hierarchy of communications within the network, the management of the hierarchy, and fault tolerance considerations. Space communication problems are then discussed. Suitable applications are briefly introduced with the paper focusing on the missile detection-destruction problem in a VLAN environment.

2. The VLAN Model

2.1 Model Design Issues

When designing the VLAN model, many issues must be considered. There are design questions about information flow, control knowledge distribution, and domain knowledge distribution. While the flow of information is not dependent on a particular problem, it is dependent on the communication plan of the network. Control knowledge can be handled in a general way in the knowledge base, but content and distribution of domain knowledge is problem specific. Should communication and knowledge be fully centralized or fully distributed? Should knowledge be redundant at nodes? This paper contains some preliminary results, but the full-blown knowledge base will be done in future work.

2.2 Organization of the VLAN Model

The complexity of a network of this size is so great that a fully centralized design must be ruled out. Similarly, a fully distributed design is also unreasonable as it would take too long to find and pass information. Thus, a trade-off between a fully centralized and fully distributed design is adopted.

The model design is described here, but has been altered from the original model found in [5] so that it is adaptable to the applications considered here. For a discussion on communication protocols for VLANs, see [3]. A general VLAN has no restrictions on the number of nodes or their positions, physically or within the network. There are four general kinds of nodes:

1. Fixed nodes stationed on earth.
2. Mobile earth nodes -- land, water, air vehicles.
3. Geosynchronous satellites.
4. Non-geosynchronous satellites.

Given the different kinds of nodes in the network and the possibly very different tasks that these nodes will accomplish, nodes are grouped together functionally based on tasks. In the general model, it is assumed that each node is responsible for a set of very specific tasks as well as some general support. Thus, each node is assigned membership in one or more functional group based on its responsibilities. The task may be some specific computational need or some managerial function. These groups are totally functional, and not physical.

2.2.1 Communication Hierarchy

Because the physical distribution of nodes could differ greatly from the functional distribution of nodes, direct communication with each other may be impossible. So in addition to the functional groupings, the network nodes are also assigned to a physically localized group, called a "partition". That is, physical space is divided into partitions made up of nodes located physically in the same proximity. Within each partition are nodes from some functional group, physically splitting the functional groups into parts. Therefore a partition is comprised of possibly many functional subgroups which will be referred to as local functional groups. Each node is a member of exactly one physical partition, but may be a member of more than one functional group. A sample VLAN, illustrating three physical partitions and two functional groups is shown in figure 1. The naming, subscripting, and superscripting scheme explanation follows.

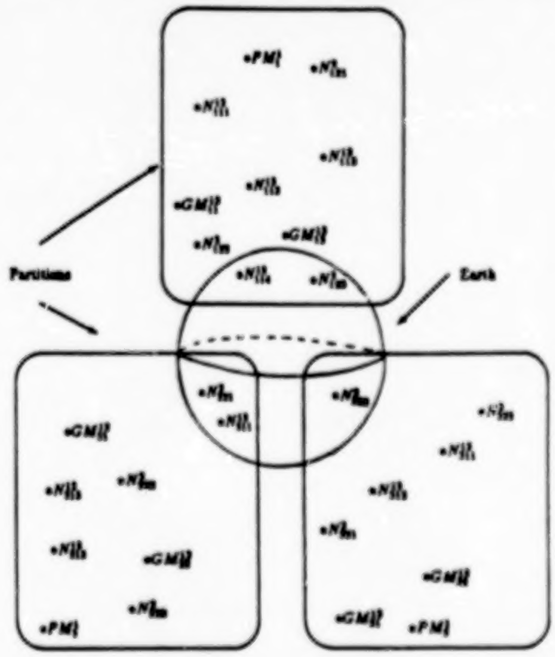


Figure 1: *VLAN* with 3 Partitions and 2 Functional Groups

Each node in a partition is called a "cell" node. In figures, cell nodes are labeled with an N. For communication purposes, each cell node is part of one local functional group. One of the cell nodes is chosen as the master node per local functional group in each partition, and is called a "group master node" or just Group Master (GM) and is labeled in figures by a GM. Thus, each partition contains one group master for each local functional group represented in that partition. Again within the partition, one of the cell nodes is given the duty of master over the group masters and is called the "Partition Master" (PM) and is labeled in figures by a PM.

Thus, each partition has exactly one partition master which communicates with group masters. Each local functional group has exactly one group master which communicates with cell nodes in that group. This forms a hierarchy of communication from PMs to GMs to cell nodes. To correspond from one partition to another, partition masters communicate. The subscripting and superscripting scheme in figures is as follows. For all nodes -- PMs, GMs, and cell nodes -- all digits in the superscript say which functional groups the node belongs to. PMs have one subscript, simply taken from a sequential numbering of all partitions and representing their respective partition number. GMs have two subscripts, the first adopts the subscript of their PM indicating their partition number, and a second subscript represents the local functional group the GM is a part of. Similarly, cell nodes have three subscripts, saying which partition and which local functional group the cell node is in and then an identifying number within the local group. Since the superscript lists all functional groups the node is a member of, the second subscript of group masters and cell nodes must be contained in that list. A representation of the communication hierarchy with two partitions is found in figure 2.

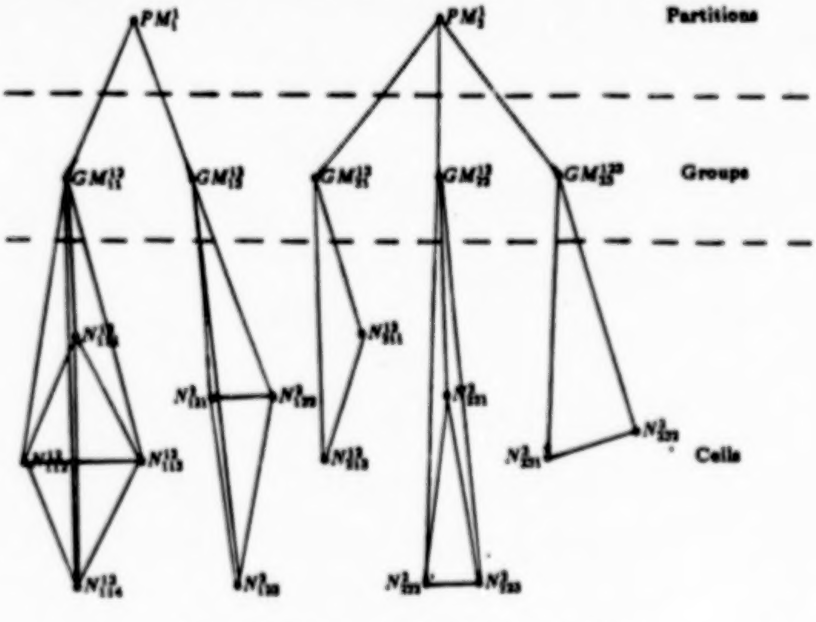


Figure 2: Communication Hierarchy of a *VLAN*

2.2.2 Management Hierarchy

The last section introduced the communication hierarchy of the network, and now the management issues are addressed. At a high level, a group master manages all operations of its group's cell nodes, and is the communication link between the cell nodes and its partition master.

A partition master coordinates communications between local functional groups in its partition and communicates with all other partition masters. It is assumed that partition masters will be on board geosynchronous satellites. There is less interference for satellite to satellite communication than with that of earth to earth or satellite to earth communications. An earth node is designated as a back-up or stand-by partition master to assure continuation of the system in case of failure. Because mobile earth nodes are not fixed and their location can be readily changed so that they are not predictably located, this node type will take on the important role of back-up node.

In the general model, management of direct communication takes place as follows:

> A cell node may initiate direct communication with its group master, or directly with another cell node in its local functional group only if initiated by its group master.
>
> A group master may initiate direct communications with any cell node in its localized functional group, or with its partition master.
>
> A partition master may initiate direct communications with a group master in its partition, or another partition master.
>
> Under special circumstances (eg., failure of a supervisory group master), a cell node may initiate direct communications with its partition master to notify of the failure. The partition master may then initiate direct communications with a cell node to promote it to group master status.
>
> Similarly, a group master may initiate direct communications with another partition master to notify it of failure.

2.2.3 Fault Tolerance

In the general model, when a cell node "determines" that its group master has failed, it directly informs that group master's partition master of the suspected failure. After "verifying" the failure, the partition master appoints one of the remaining cell nodes in the functional group as the new group master. The knowledge base is updated.

A similar dynamic fault tolerance scheme exists for partition master failures. A group master determines its partition master has failed. If the partition master is the original satellite node which has failed, then its earth duplicate is notified and after checking to make sure of definite failure, either it assumes partition master duties or assigns the duties to another geosynchronous satellite.

3. Space Communication Problems

3.1 Suitable Problems

The VLAN model design lends itself well for use with physically large distributed problems. Since most space communication problems fall into this category, the VLAN model is ideal to use as a framework to handle these problems. It is organized in such a way that a complex management system can be neatly imbedded in the network.

Many distributed artificial intelligence systems have been developed over the years in such areas as medical diagnosis, natural language processing, and manufacturing. (See [1] for a concise survey.) The VLAN is inappropriate for these smaller problems, but is a way to handle the physically larger problems.

One suitable application is space traffic control, the problem consisting of an airspace of space stations with spacecrafts arriving and leaving at fixed entry and exit points. Another problem is the vehicle monitoring problem. Here, a map is to be created from sensors picking up signals from or sounds of moving vehicles in space. The VLAN could be used for the business application of foreign exchange trading, communicating information around the world in real-time. Many military problems involve space with one foremost problem being the missile detection-destruction problem. The rest of the paper focuses on this problem.

3.2 The Missile Detection-Destruction Problem

One of the most interesting, complex problems suitable for the VLAN model is the missile problem. Simply put, it is the problem of detecting an incoming missile, computing it's trajectory, and ultimately destroying the missile. The network must carry out the essential functions of surveillance, trajectory determination, discrimination and assessment; aiming and tracking of nodes; interception and destruction; and management. The system must perform tasks in a fast and efficient manner and be fault tolerant, continuing functioning for as long as possible after any node failure.

The system is designed to intercept a missile in all four phases of its flight: the boost, post-boost, midcourse and terminal phases [4]. The boost phase is brief, lasting approximately 50-300 seconds and occurring 200 km above the earth's surface. The boost phase is followed by the post-boost or bussing phase. In this phase the bus (last stage of the missile) is already in a trajectory, and releases many reentry vehicles and decoys, lasting 5 minutes. In the midcourse phase, now 1000 km above the earth, the warheads and decoys are traveling in an unpowered trajectory. The midcourse phase lasts 20 minutes. In the terminal phase, a time of one to several minutes, the atmosphere is reentered and the decoys are stripped away by air resistance. The four phases are illustrated in Figure 3.

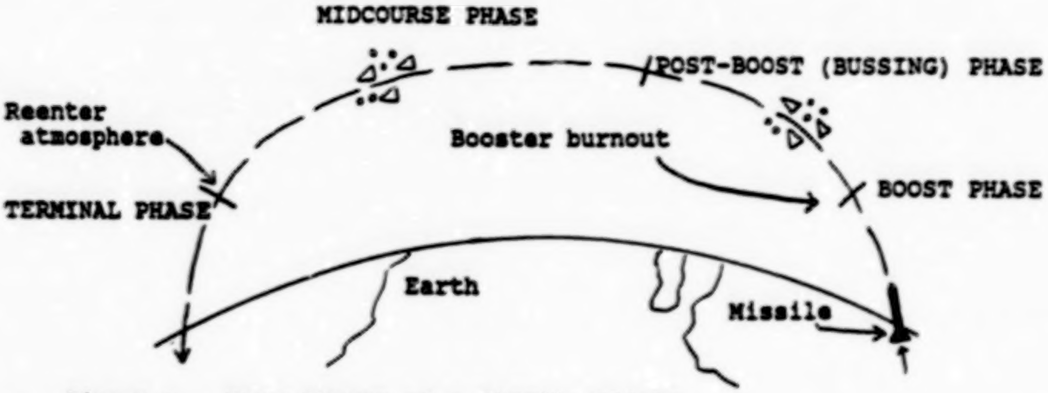


Figure 3. Four Phases of a Missile Flight.

3.2.1 The VLAN Functional Groups and Partition Structure

The first consideration of incorporating the missile problem into the VLAN model is to determine the functional groups. The needs of the missile problem dictate most of the functions.

At the crux of the problem are the functions of missile detection and destruction. The function of detection includes sending a warning to appropriate nodes or group managers, determining the source of the missile, and anticipating the projected target. The warning is to alert group managers that computation and destruction nodes under their supervision are about to be invoked. This gives them a head-start in figuring out which node should be chosen to destroy the missile. The source of the missile and the target are useful pieces of information for anticipating future missile onsets. Since this information must be kept in the knowledge-base associated with the VLAN, messages of information must be sent to the record keeping nodes.

When a missile is detected, its trajectory must be computed for use by destruction nodes. Thus trajectory computation is a necessary, important function needing fast, dedicated computers. This function will be divided into subfunctions depending on the phases of a missile's flight time, the boost, post-boost, midcourse, and terminal phases. There are also four functions associated with destruction corresponding with the four phases. Different kinds of nodes are necessary for the different phases.

Other functions include the detection of failing nodes and destroyed nodes and the updating of the knowledge-base.

Successful destruction of missiles is useful information for guidance of destruction of future missiles. If technologically possible, a decoy-warhead discriminating function could be incorporated in the VLAN. Some nodes will be dedicated strictly as protection nodes, protecting other, critical nodes. Acting as a back-up node constitutes a functional group, as well as the function of decoy satellites. In addition, there will be many kinds of management nodes. These handle the duties of keeping and controlling the knowledge-base and managing the VLAN network.

To be able to approach real time response, the VLAN hierarchy is necessary for handling the complicated duties of management. For example, if a missile is detected, the detection node communicates with its group master who either finds nodes under its command to finish the destruction, contacts other group master nodes, or contacts its partition master. There are fewer communication links to accomplish the task and they can be done in parallel to assure completion. The hierarchy is based on an information need rather than standard network traffic.

All the functional groups will be represented in each partition. The number of nodes representing some functional groups will be based on the number of nodes from some key groups. The key groups include missile detection, missile destruction, and management nodes. The exact numbers for these key nodes are not yet determined, but there have been estimates made for similar problems, see [2] and [4]. One difficulty of the VLAN model, common to all space network systems, is the changing configuration of the partition structure. Unless a satellite is in geosynchronous orbit, it will not be stationary above a point on the earth, but will trace a track which rotates around the earth. There must be enough satellites in each partition so there are always necessary nodes within intercept range of any missile.

Numbers of trajectory computation, node failing detection, and decoy discrimination nodes will depend on the number of missile detection nodes. The amount of missile destruction nodes will determine quantities of node failing detection nodes as well and destruction detection. Protection and back-up nodes will mostly depend on how many management nodes there are. Decoy satellites are not dependent on any particular key nodes, and are not key nodes themselves, but are dependent on the total number of nodes in a partition. Their numbers will be in a proportion so as to make it difficult to determine the real nodes.

3.2.2 Problem Knowledge

The domain knowledge of the missile problem includes many pieces, the most important being the topology of the VLAN, that is, the partition division and the numbers and positions of nodes. This information is distributed amongst the management nodes and their back-up nodes. When nodes dynamically fail, the news of the failure is broadcast to all group managers.

4. Conclusions

This paper introduces a very large area network as a framework for solving spatially large distributed problems. A VLAN is composed of cell nodes, group masters, and partition masters with the partition masters managing the group masters in their partitions, and the group masters managing the cell nodes in their localized groups. All nodes belong to some functional group based on the tasks they perform. The key elements of the model design are the communication and management hierarchical organization, with the motivation for the hierarchies to simplify the complexity of the network. The VLAN model is designed to be fault tolerant for as long as possible.

This distributed approach to solving problems is powerful with much potential for solving futuristic distributed problems. The VLAN model with its organization is well suited as an environment for handling these problems. Several possible problems were introduced with a focus on one foremost problem, the missile detection-destruction problem. This problem was outlined and incorporated into the VLAN model.

There is still much work to complete the incorporation of the missile problem into the VLAN model. Domain knowledge needs more detail and precise placement in the model and control knowledge must be totally defined and distributed in the network. The preliminary employment of the VLAN model for use with distributed problems is promising.

References

[1] Decker K. S., Distributed Problem-Solving Techniques: A Survey, IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-17, No. 5, Sept/Oct 1987, pp. 729-740.

[2] Guertner G. L. and Snow D. H., The Last Frontier, D. C. Heath and Company, Lexington, Massachusetts, 1986.

[3] Oliver S. R. and Wolf J. J., Characterizing Very Large Networks (VLAN), Proceedings of Computer Networking Symposium, Addendum, Oct 1986.

[4] Schroeer D., Directed-Energy Weapons and Strategic Defence: A Primer, Halstan & Company Ltd., Amersham, Bucks, Great Britain, Adelphi Paper 221, 1987.

[5] Wolf J. J. and Ghosh B., Modeling Very Large Area Networks (VLAN) using an Information Flow Approach, IEEE Proceedings of the Symposium on the Simulation of Computer Networks, Colorado Springs, Colorado, 1987.

N89-15602

Ada in AI or AI in Ada? On Developing A Rationale For Integration

Philippe E. Collard
California Space Institute
A-016, UCSD, La Jolla, Ca 92093

Andre Goforth
NASA/Ames Research Center
Information Sciences Division, MS244-4
Moffett Field, Ca 94035

Abstract

The use of Ada as an Artificial Intelligence (AI) language has been gaining interest within the NASA community. This interest is held by parties in NASA who have a need to deploy Knowledge Based-Systems (KBS) compatible with the use of Ada as the software standard for the Space Station.

A fair number of KBS and pseudo-KBS implementations in Ada exist today. Currently, no widely used guidelines exist to compare and evaluate these with one another. The lack of such guidelines illustrates a fundamental problem inherent in trying to compare and evaluate implementations of any sort in languages that are procedural or imperative in style, such as Ada, with those in languages that are functional in style, such as Lisp.

This paper discusses the strengths and weakness of using Ada as an AI language and provides a preliminary analysis of factors needed for the development of criteria for the integration of these two families of languages and the environments in which they are implemented.

The intent for developing such criteria is to have a logical rationale that may be used to guide the development of Ada tools and methodology to support KBS requirements, and to identify those AI technology components that may most readily and effectively be deployed in Ada versus those best left in a functional language.

INTRODUCTION

The AI community and the Ada software engineering community have historically evolved and developed methodologies and formalisms with only a modest amount of collaboration to date. This is partly due to their respective goals. On one hand, the AI community has been concerned with the "what" and "why" of a problem, *i.e.*, the representation of information indigenous to the development of requirements. On the other hand, the Ada software engineering community has concerned itself more with the "how to" or methodology of correctly arriving at a system implementation that may have little in common with the knowledge used in generating the original requirements.

By mandate, NASA has made Ada the *de facto* language for the Space Station. The authors believe this decision was rightfully made on the grounds that Ada and the software engineering principles it embodies will be an invaluable assets in keeping development and maintenance costs manageable at all levels. In the past ten years, Ada has made significant progress in realizing the expectations of its designers. There is merit, however, in the intuitive, and often demonstrated, notion that there is no "silver bullet" [1] in computer sciences. There are and probably will continue to be, difficulties in using a single tool for the implementation of all the components of Space Station software, be they AI systems or not.

A major issue confronting those parties who have a need to deploy both ground and in-flight KBS systems is: *how is this mandate to be interpreted or executed?* For example, policy makers may view the mandate as allowing for three alternatives in hosting AI technology in the Space Station software environment:

- Ada as the only host language. In this alternative, regardless of any other considerations, Ada is the only allowable host environment. Any AI technology to be used in the Space Station must, therefore, be made to operate in Ada.
- Ada as a co-host language. Common Lisp (and other desirable AI languages) is accommodated in the Space Station environment with Ada through the use of standard interfaces and the implementation of common development tools and maintenance procedures.
- Ada as a temporary co-host language. Common Lisp is accommodated as described above, but with the condition that, over a certain period of time, all non-Ada code must gradually be phased into an Ada implementation. There may be other variations of this case that may be of interest to policy makers and user advocates.

The mandate interpretation issue applies to implementors and users of other specialized computer technology areas that are perceived as being outside the current capability envelope of Ada implementation. For example, real-time requirements of some subsystems and the database search and retrieval requirements of some applications may have non-Ada "off the shelf" solutions which come with lower risk factors for meeting a required capability, resource, and schedule envelope. However, in this case, the risk factor re-surfaces in the effort required to integrate , verify and maintain the "non-native" solution with the host Ada environment.

The dilemma of local or project-specific optimization, in terms of use of a special technology, versus the strategic or global system optimization, in terms of a mandated technology choice, is inevitable. The matter of making exceptions, *i.e.* waivers, to a policy might be viewed as detrimental to the intent and execution of the policy. However, it may be argued that some compromise is necessary in these circumstances. The general nature of the problem is brought out here because a solution to it will have an impact on the solution to the specific problem at hand; *i.e.*. how to accommodate AI technology and "culture" in the Space Station software environment.

A recommendation for resolving this dilemma is beyond the scope of this paper. The scope is limited to discussing the technical issues of accommodating AI technology and "culture" in Ada from a managerial view. The purpose of this discussion is to enhance the level of understanding of the issues and factors that need to be explored more thoroughly for developing effective integration criteria for Ada as an AI language in the Space Station.

The advantages of Ada for the implementation of Space Station AI systems are reviewed in the following discussion.

AI IN ADA: STRENGTHS

a) Ada is a standard

Ada is now an ANSI and ISO standard. Its definition is clearly stated by the Language Reference Manual (LRM). Unfortunately, the validation suite does not address performance issues, but it does ensure that an Ada compiler delivers objects that conform to the syntax and semantics defined by the LRM. If some latitude is given to the implementors by Chapter 13 of the LRM, the variations from compiler to compiler are limited to specific features, and must be clearly documented. All in all, the availability of such a standard definition allows for a smoother development cycle by increasing the portability and re-usability of software components, facilitating maintenance, and reducing the training cost. All of these things will benefit the development of the Space Station AI applications and their interfacing with other components of the Space Station software.

Standard definitions for AI languages, even the most commonly used such as Lisp and Prolog, are not as well developed as for Ada, though efforts are underway to unify the various dialects of Lisp into Common Lisp. For Prolog, the definition given by Clocksin and Mellish [2] is the closest to what may be called a widely accepted standard.

This lack of standardization can only be detrimental to the development of large, "real life" projects such as the Space Station. This conclusion, reached by the DoD community after studying the consequences of the proliferation of programming languages and dialects providing a rationale for the development of Ada [3]. Having a standard that is certifiable by an independent means is very important for gaining Agency acceptance.

b) Ada favors the use of modern software engineering techniques

Ada is not only a programming language but also a methodology of development for large software projects. Ada was designed for "programming in the large". By contrast, software engineering has never been a predominant concern in the AI community. One reason for that is the fact that the implementation of AI systems has traditionally been based on rapid prototyping and incremental development - initially an appropriate approach because it was well suited to the size of the applications being developed. As stated in the introduction, software engineers focus on the "how to", whereas knowledge engineers focus on "what" and "why". The two groups use different tools and methodology.

Some may argue that the concept of "engineering" is contrary to the dynamic nature of AI systems. After all, these systems exhibit features, such as self-modifying code, that are considered harmful in light of the current software engineering principles. On the contrary, it has been demonstrated that AI work and software engineering are, indeed, complementary and not orthogonal [4].

To reach today's expectations, Space Station AI systems will have to be of a size and scope that have no current equal. Furthermore, they will have to be highly reliable and maintainable. It may be questioned whether existing iterative processes used to develop today's AI systems can be directly scaled to accommodate the increased size and scope. The rationale for questioning and caution is based on the past experience in the AI field that AI techniques, in some cases, have not scaled-up effectively. Incorporating the experience and methodology of the Ada community may greatly aid in the scaling-up of AI systems in the Space Station.

c) Ada is available on space qualified hardware/software platforms

Because of its special sponsorship by the U.S. Government, Ada is becoming available on an increasing number of hardware and software platforms suitable for space flight. Because of its standard definition, applications developed on one of these space-qualified platforms can easily be ported to another platform, provided they avoid making references to system-dependant features (as defined by Chapter 13 of the LRM). This allows for greater flexibility in the development process.

d) Ada's tasking construct is well suited to AI applications

Although Ada was not initially designed for AI, it contains some constructs that are well suited or readily adapted to AI work. The most notable example is tasking, giving Ada the ability to support parallel processing at the language level. Whether it is for fine-grained parallelism [5] or for large-grained parallelism [6], the availability of the tasking construct allows for the development of AI systems in terms of concurrent execution of computing or inferring units without requiring any radical extension or modification of the language.

e) The use of Ada can improve the development of real-time expert systems

In the context of the Space Station, on-board AI applications will have to respond in real-time or near real-time. The status of real-time expert systems is far from being adequate. A survey of this field recently published in AI Magazine[7] , a publication of the AAAI, concluded that: "current expert system shells are two or three orders of magnitude too slow", "current shells cannot guarantee response times", "current shells have little or no capabilities for temporal reasoning", "current shells lack the facilities to handle hardware and software interrupts". All these features, currently lacking in available AI tools, are critical to the implementation of Space Station systems.

Real-time expert systems is another area where the use of Ada may be extremely positive. Ada was designed specifically for embedded real-time systems. Although the suitability of Ada real-time constructs is open to debate, it is undeniable that the use of Ada for real-time systems is responsible for major advances in this field. First, real-time systems can be designed and coded more cleanly using software engineering techniques. Second, the language supports real-time features such as tasking and exceptions and hardware interrupt handling without extension or references to any particular operating system. Lastly, significant progress is being made in the area of run-time systems to support Ada on embedded targets. Thus, one of the consequences of using Ada for building real-time expert systems is benefit from these advances and, therefore, a remedy some of the problems encountered with current AI tools.

f) Ada may be used as a standard PDL for AI applications

Even if Ada was not used for the implementation of some Space Station AI systems or subsystems, it could be used as Program Design Language for these same systems. This will ensure a commonality in the communication media between the various development teams and therefore would facilitate interfacing and integration.

The following discussion reviews the limitations of Ada with regard to the implementation of Space Station AI systems.

AI IN ADA: WEAKNESSES

a) Rapid prototyping is difficult in Ada

Rapid prototyping is a trademark of AI development and requires tools with great flexibility. There are two major drawbacks to the use of Ada for building prototypes of AI applications. First, Ada is a compiled language. The LRM defines rather strict rules regarding what is to be re-compiled when a unit of the compilation library is modified. These rules may imply that large portions of a system may have to be re-compiled after just one modification. This puts a heavy burden on the prototyping work. Unlike interpreters, compilers are not well adapted to interactive development. They are designed to produce efficient object modules, whereas interpreters are designed to allow quick and easy modification and testing of programs. However, the development of incremental compilers may help reduce the overhead imposed by compilations and re-compilations.

The strong typing rules of Ada constitute the second obstacle to using Ada for AI prototyping. With Ada, every variable must be declared of a certain type. The iterative and dynamic nature of the prototyping work would be likely to cause frequent changes in the typing scheme, requiring type declarations to be traced and modified throughout the program. This could add significantly to the overhead as well as multiply the causes for error.

b) Technical difficulties

There are a number of linguistic technical problems that limit the use of Ada for AI applications. Depending on the specificity of each application, the impact of each is felt differently. For example, it is much easier to use Ada for developing an expert system shell than for doing symbolic computation. Some of the major technical problems include:

1) Storage management and lack of garbage collection. AI applications are prone to high consumption of dynamic storage. The definition of Ada does not require an implementor to provide garbage collection facilities. In addition, Ada does not allow for static pointers (pointers to objects that are already declared).

2) Early binding. Binding decisions are made early in Ada. This is not the case for most AI languages. Early binding reduces, and sometimes prevents, the possibility of dynamically defining new types as required by the evolving context of the execution (for instance, by creating new data types at run-time).

3) Functions and procedures cannot be arguments of other functions or procedures. This prevents self-modifying code from being implemented in Ada. Therefore, it is more difficult for an AI application written in Ada to adapt to a changing context than for the same application written in Lisp. More generally, Ada lacks the dynamic constructs that are commonly found in most AI languages.

c) Cultural and training issues

G. Booch, a noted Ada expert, noted that "a programming language shapes the way we think about a solution. We need a language that leads to systems that map directly to their problem space" [8]. This is where "cultural" differences appear between the AI community and the software engineering community. One concerns understanding the theory of computation, the other concerns building efficient and reliable computational engines. This has lead to a divergence in design and evolution of the tools used by each group. Using Ada for developing AI systems implies that each community will have to understand to a certain extent and, adopt the cultural background of the other to merge it with its own.

A NEED FOR ADDITIONAL FACTORS

Developing a list of Ada's strengths and weaknesses as a development and implementation environment for AI is a step toward formulating a rationale for integration. A more extensive and exhaustive study may uncover additional strengths and weaknesses. Weights could then be assigned to each of these, according to some priority scheme, and the choice made, based on the relative total weights. This, however, provides an incomplete picture. First, given such a short list where every point is pivotal, arriving at a rational weighting scheme may prove to be intractable or unacceptably arbitrary. Second, the possibility of Ada as a co-host is, for the most part, overlooked since this could only be covered by adopting an arbitrary scoring range to represent it as a choice. Therefore, additional factors may have to be considered in developing an integration rationale.

An additional factor to consider is the development of evaluation criteria composed of significant software engineering factors. A list of such factors, not to be construed as exhaustive, is given below:

- Project Size For large scale projects, one must evaluate which solution provides the best way to handle all the problems related to size.

- Training Is there a readily available pool of trained engineers who can make the solution work for all aspects of the projects?

- Performance What is the solution that is most likely to provide better performance in the short- , mid-, and long-term?

- Interfacing to other environments Which solution will be the easiest to interface to other systems, subsystems, or environments ?

- Evolution Which solution will provide a smoother evolutionary path?

- Maintenance Which solution will facilitate maintenance best?

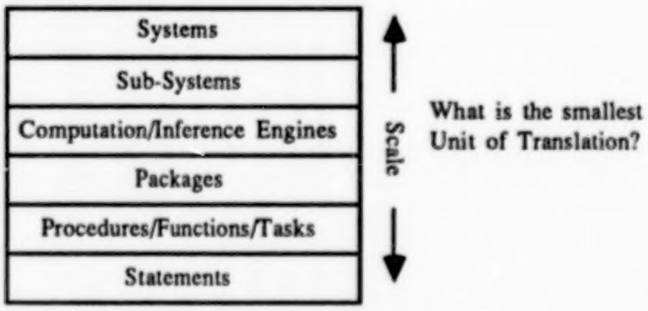
- Verification and Validation Is there a solution that makes Verification and Validation easier to perform ?

- Productivity tools Which solution provides the better software productivity tools and life cycle environment?

- Real-time accommodation For applications that require real-time or near real-time performance, which solution is likely to perform better?

Other factors to consider are the various approaches to using Ada as an AI language. Emulation of Lisp features in Ada with special packages and techniques is one possibility. To better understand these and other approaches, it is useful to attempt to categorize them in some uniform and constructive manner. Any attempt to replicate the behavior of an application written in Lisp in Ada depends on the scale at which a translation is performed. The scale is illustrated in the figure below as a hierarchy of levels in the Space Station software environment.

Space Station Software Environment Levels


Systems
Sub-Systems
Computation/Inference Engines
Packages
Procedures/Functions/Tasks
Statements
Scale
What is the smallest Unit of Translation?

The figure above shows the different levels at which the insertion of AI technology may be made in the Space Station software environment. Any approach to implementing AI technology must enter at some level of the hierarchy, though it may cover more than one level. Since there is no accepted or standardized way to categorize all of the approaches for using Ada as an AI language, the following categorization is put forward based on information currently available in the literature:

- Automatic Translation to Host Environment
- Emulation of features in Host Environment
- Interface/Front-end Processing to Host Environment
- Host Implementation From Requirements Specification

The boundaries between these are not sharp, and some AI in Ada approaches may be viewed as hybrids of these. The issues of the suitability of using Ada for AI was raised early in the life of Ada[9]; however, the topic has not been explored as nearly as much as other Ada issues, such as, real-time support. Still, the literature available on the topic allows for a preliminary assessment of the approaches. In all likelihood, the research and development of this Ada topic will burgeon enormously in the very near future. A number of the publications available to date are listed in the references for this paper [10],[11],[12],[13],[14]. In addition, under the auspices of George Mason University, Fairfax Va., in cooperation with the Software Productivity Consortium, the AIDA conference is held annually and serves as a forum for parties interested in studying Ada and Artificial Intelligence.

The most straightforward way to use Ada as an AI language is to use an automatic translator. Lisp code is input to a translator that produces Ada code which is then compiled and executed. Ideally, the results of the Ada version are identical to those found in the Lisp version.

At present, the automatic translation approach is being applied primarily to the Statements layer. Due to the large semantic gaps between Lisp, a functional language, and Ada, a procedural language, this approach has, so far, been limited to restricted subsets of Lisp. The linguistic issues have been discussed in the literature[12],[15].

Emulation of Lisp features in Ada is closely allied with translation at the Statements Level. Although this approach enters at a higher level of the hierarchy than the automatic translation approach, the boundary between the two is not sharp. This approach may cover the next two higher levels in the hierarchy, the Procedures/Functions/Tasks and Packages. With this approach, libraries of packages are provided to the Ada programmer with which to fashion the emulation of many list processing types of constructions. One such approach was reported by Reeker and Wauchope in 1986 [16].

Interface, or front-end processing is directed mainly at the Sub-systems and Computation/Inference Engines level. Here too, the boundaries are not sharp. In this type of approach the AI user is presented with a familiar expert systems shell format that is implemented all or partly in a non-Ada language and environment. The results are then processed into the Ada environment. In this approach, the critical link is the specification of the interface that performs the conversion/translation of knowledge-based representation into Ada data types.

Host implementation from requirements specification is the highest level of all the approaches in that it is directed at the Systems and Sub-systems level. A requirements specification of an AI application, such as an expert system shell, is generated and then used to implement, from scratch, a working version in Ada.

In any analysis of the relative strengths and weaknesses of these approaches, one overriding element that may be difficult for either side to fully appreciate is "culture." In the context of Ada and AI, culture may be defined as the priorities, the *raison d'etre* or collective principals, that serve to guide the constituents in how they accomplish their goal(s). Many capabilities that cannot be merely identified as a piece of code are potentially lost in performing any one of these approaches of using Ada as an AI language. As mentioned earlier, rapid prototyping is a hallmark of the AI community. Yet, it is difficult to convey to someone who is familiar only with Ada and its software engineering principals just what it all means, other than it is possible to accomplish a prototype and, possibly, a full scale development in a certain (record breaking) time.

Due to the necessary limitations on the length of this paper, discussion of the application of the system engineering factors, mentioned previously, to each of these approaches is limited to discussing only those criteria salient for the case of automatic translation. This discussion illustrates another step in developing an integration criteria.

Currently, commercial products exist wherein Common Lisp is compiled into C and then this code is compiled into machine code to be run on the desired host. However, there are software engineering issues that are not fully addressed with this approach even if a translator could be built that would be capable of *merely* translating correctly all of the features and language constructs occurring in Common Lisp into Ada code. They are as follows:

- Performance. Whether the application code translated into Ada should perform as fast or as slow as its implementation in Lisp ?

- Real-time Accommodation. For translation of real-time codes it appears highly unlikely that such a translator could insure the same real-time characteristics found in the Lisp version in the Ada version.

- Evolution. If new features are incorporated in the Common Lisp standard, then the translator must be updated and reverified. This task may be equal in cost to the one of building the original translator.

- Maintenance. The Ada code compiled by the translator is most likely indecipherable by software engineers unless it is built to provide comments on the translation process. Indeed, this may be equal in difficulty to building a translator that merely translates. Thereby the cost is doubled. As a consequence, all Ada code generated by the translator without built-in commentary will be dependent on the translation step. That is, any changes due to updates or due to fixing errors in the Ada version will require going back to the Lisp version of the code and making the change there, then verifying the correctness there, and then performing the translation. Therefore, the maintenance cycle for codes translated mechanically will always be hostage to the translator and the Lisp implementation of the code.

These engineering issues are less of a problem if Ada is the "native" or systems programming language of the host machine. An example of this occurs with the programming language C in which 95 percent or more of the operating system UNIX is written. The case of Ada being the systems programming language for a operating system on the scale of UNIX is a question of availability. Implementing a general purpose operating system in Ada and translating Common Lisp into efficient Ada code are technology areas that appear to need more development.

SUMMARY

Some of the factors needed in developing an integration criteria have been discussed in this paper. No conclusions are to be drawn from the discussion because the relative importance of many of the factors is open to yet another degree of scrutiny. Even if a concerted effort was mounted to once and for all answer what is the best choice - there are possibly several equally good alternatives - it is highly unlikely that it would adequately reflect the needs of all the interested parties and the evolutionary processes rapidly at work in both the Artificial Intelligence community and the Ada community. The one common denominator necessary in whatever choice is made is *training*.

There are currently relatively few people trained in both Ada software engineering and AI. This is a significant problem for the design and implementation of the Knowledge-Based Systems that are envisioned for the Space Station. For "AI in Ada" or "Ada in AI" to flourish, Ada software engineers will have to understand knowledge engineering techniques and their rapidly evolving definitions. AI specialists will have to take into account the requirements imposed by large, long-lived projects where definitions and techniques must be stabilized early in the development process. With this cross-cultural understanding, the question "Ada in AI or AI in Ada? " will become a moot point.

REFERENCES

[1] F. Brooks, No Silver Bullet: Essence and Accidents of Software Engineering, IEEE Computer, April 1987

[2] W.F. Clocksin and C.S. Mellish, Programming in Prolog, Springer-Verlag, 1981

[3] Requirements for High Order Programming Languages, IRONMAN, Department of Defense, January 1977

[4] G. Karam, An Icon-Based Design Method for Prolog, IEEE Software, July 1988

[5] A. Brintzehoff, S. Christensen, J. Mangan, J. Greco, The Use of Ada Concurrent Processing Features in an Implementation of Parallel Tree Searching Algorithms, Proceedings of AIDA-87, Third Annual Conference on Artificial Intelligence and Ada, Washington D.C., 1987

[6] R. Volz, P. Krishnan, R. Theriault, An Approach to Distributed Execution of Ada Programs, NASA Workshop on Telerobotics, January 1987

[7] T. Laffey, P. Cox & al., Real-Time Knowledge Based Systems, AI Magazine, Spring 1988

[8] G. Booch, Software Engineering with Ada, 2nd ed., Benjamin Cummings, 1987

[9] R. Schwartz, P. Melliar-Smith, On the suitability of Ada for Artificial Intelligence Applications, Project 1019, SRI, 1980

[10] P.J. Wallis, Automatic Language Conversion and its Place in the Transition to Ada, Proceedings of the Ada International Conference, Paris, 1985

[11] A. Rude, Translating A Research Lisp Prototype to A Formal Ada Design Prototype, Proceedings of the Washington Ada Symposium, 1985

[12] P. Bhugra, T.N. Mudge, Comparisons between Ada and LISP, U. Michigan, Research Report, 1985

[13] S. Reddy, F. Van Scoy, Knowledge Representation in Ada, Proceedings of the Eastern Simulation Conference, Orlando, 1987

[14] K. Warn, Lisp vs. Ada Implications in Diagnostics Oriented Expert Systems, Proceedings of AUTOTESTCON, 1986

[15] Terry B. Bollinger, Ada and PROLOG - A Few Observations, Proceedings of AIDA-87, Third Annual Conference on Artificial Intelligence and Ada, Washington D.C., 1987

[16] L. H. Reeker, K. Wauchope, Pattern-Directed Processing In Ada, IEEE Computer Society 2nd International Conference on Ada Applications and Environments. April 8-10, 1986, pp 49-56.

N89-15603

Automatic Scheduling and Planning (ASAP) in Future Ground Control Systems

Sam Matlin
GE Aerospace
Valley Forge, PA, 19406

ABSTRACT

This report describes two complementary approaches to the problem of space mission planning and scheduling. The first is an Expert System or Knowledge Based System for automatically resolving most of the activity conflicts in a candidate plan. The second is an Interactive Graphics Decision Aid to assist the operator in manually resolving the residual conflicts which are beyond the scope of the Expert System. The two system designs are consistent with future ground control station activity requirements, support activity timing constraints, resource limits and activity priority guidelines.

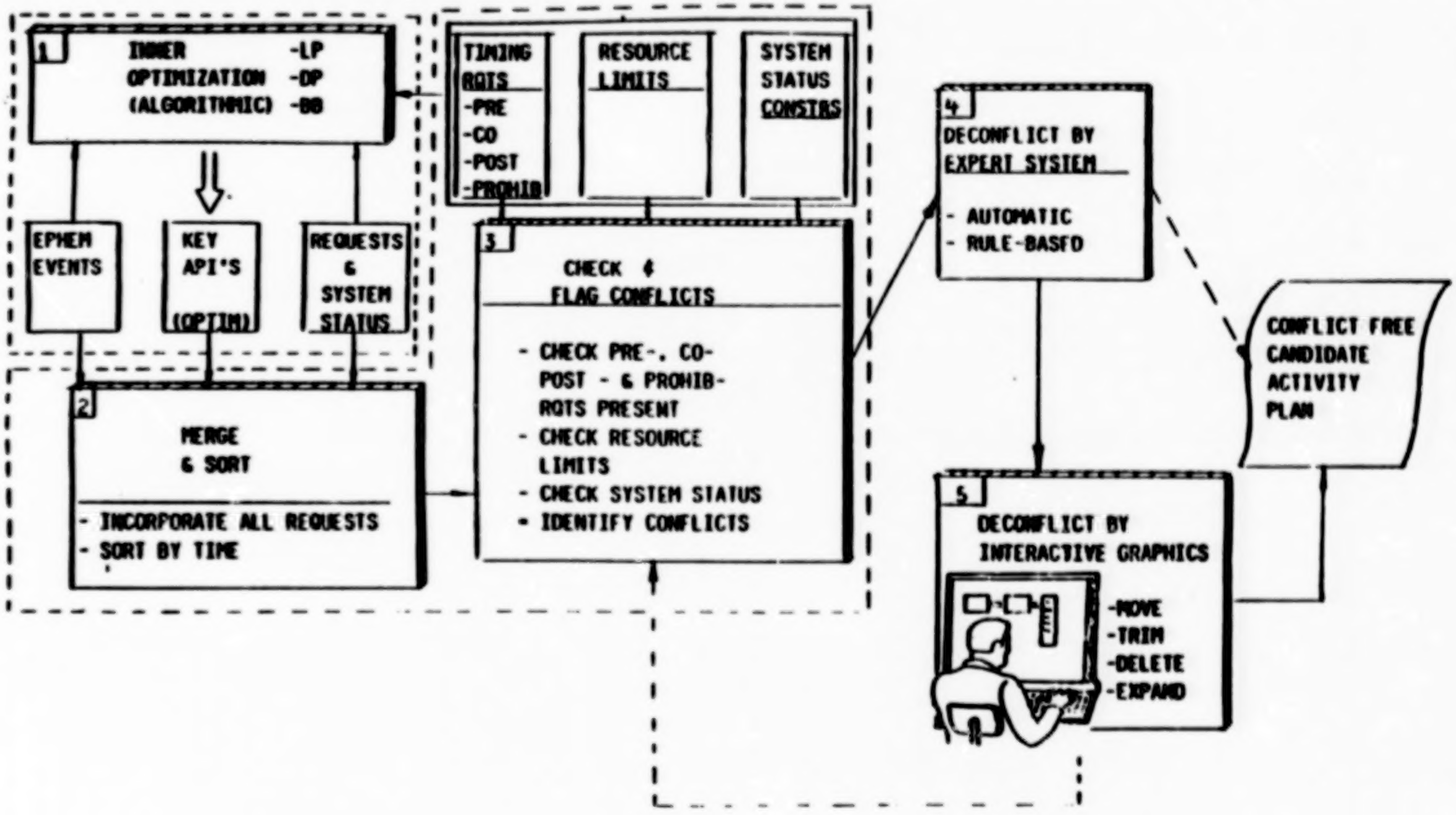
INTRODUCTION

Space mission planning and scheduling is typically performed in a labor-intensive manner, requiring significant numbers of highly skilled personnel, and limited in effectiveness by timeline constraints. By automating these repetitive labor-intensive tasks it will be possible to reduce manpower requirements and provide earlier, more reliable schedules.

Planning and scheduling has been successfully performed at GE by the procedures illustrated in Figure 1. The activities to be scheduled, referred to as Activity Planning Items or APIs, consist of such items as Key Activities (eg, Space Experiments), Special Activities (Calibration, Alignment, Test), Communication Activities (Acquisition), Supporting Activities (Housekeeping, Orbit Adjust), and others. The Key Activities are first scheduled based on a suite of mathematical optimization techniques consisting of Linear Programming, Dynamic Programming, and Branch and Bound. These Key APIs are then merged with other activity requests, and all activities are sorted by start time. Since conflicts may have been introduced by this merging of optimally scheduled activities and ad hoc or late arriving activity requests, conflict criteria (timing requirements, resource limits and system status constraints) are checked and conflicts are flagged. Typically an operator would then manually resolve these conflicts by moving or deleting activities. Instead, it is proposed that the expertise used by the operator be captured in an Expert System (ES), and that most of the conflicts be automatically resoved by the ES. Since not all conflicts are resolvable automatically (too many complex situations would have to be modeled, greatly increasing the cost, size and run time of the ES), it is further proposed to provide a decision aid for the operator in the form of an Interactive Graphics (IG) workstation to assist in resolving the residual hard conflicts.

FIGURE 1

AUTOMATIC SCHEDULING AND PLANNING



APPROACH

The following tasks were undertaken in order to achieve the study objectives of reducing operational costs and timelines:

a) Research Existing Planners. The literature contains dozens of articles on automatic planning and scheduling, including JPL's Devisor, an Artificial Intelligence planner for Voyager missions, and GE/TRW's "Automatic Mission Planning and Scheduling Expert Stystem (AMPASES)". While the literature was not directly applicable to our particular problem, useful elements and techniques were harvested.

b) Determine Application Requirements. Internal documents were reviewed and experts interviewed to ensure that the right problem was being addressed. The task was to generically characterize Key Activities, Supporting Activities, Pre-requisites, Co-requisites, Post-requisites, Prohibited Concurrent Activities, Sequence Constraints, Resource Constraints, and Priority Guidelines.

c) Develop Algorithms. Automatic Activity Planning approaches described in the literature include Expert Systems, Tree Searches, 0/1 Programming, Bin Packing, Dynamic Programming, PERT, Network Flow, and others. The effort in this task was to identify the best technique or suite of techniques to use. The conclusion was to develop an Expert System to capture the expertise of current Activity Planners. This ES was then used to remove conflicts generated by the process of accepting all requests, merging them, time-arranging them, and identifying resulting conflicts based on scheduling and conflict criteria.

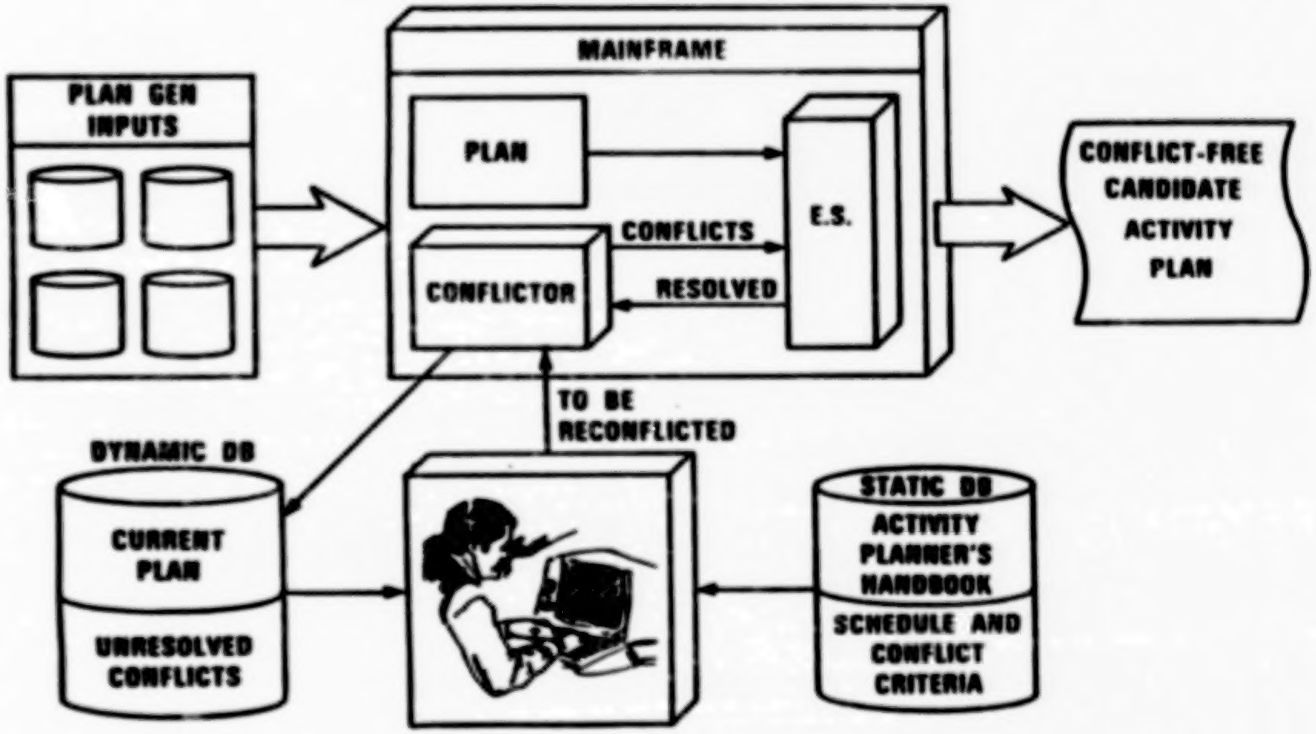
d) Interactive Graphics. Since it was not feasible to automatically resolve all the conflicts, the approach was to assist the operator with the hard remaining conflicts by providing a computer-based decision aid to facilitate this.

Two prototypes were designed to be configured as shown in Figure 2, based on the above task outputs. Note that after the ES completes its task and the operator completes his, the results of both are reconflicted to ensure that neither the ES nor the operator introduced new conflicts, and the outcome is truly conflict-free.

FIGURE 2

AUTOMATIC SCHEDULING AND PLANNING

AUTOMATIC ACTIVITY PLANNER ENVIRONMENT



RESULTS

The ES prototype was implemented with about 3000 lines of Fortran on an IBM mainframe. The knowledge base consisted of 40 'packed' rules; these are rules containing variables which can be given different values, used in conjunction with the 'Packed Rules Database' which assigns values to these variables. These 40 rules are the equivalent of several hundred ordinary rules but are more compact and more easily maintained. The rules were knowledge-engineered by consulting with several Activity Planners and Operators, based on an initial plan having 117 conflicts. The 40 rules were sufficient to resolve all 117 conflicts; they accomplished this in 1.5 seconds of CPU time, in contrast with an estimated operator time of about 30 minutes. It was anticipated that the ES,, when faced with a new plan it had not seen before, would resolve 50-75% of the conflicts with the same 40 rules. In fact it resolved 93% of the conflicts in a second plan, without introducing any new conflicts.

Figure 3 illustrates a simplified Activity Plan fragment. The conflicts are flagged in the first field, and the corresponding conflict message which the system produces is shown at the bottom. In this case activity SSSS which starts at 07:30:00 is scheduled incorrectly with respect to activity NNNN. The conflict message indicates that the type of conflict is that the required start time of SSSS was scheduled wrong - it was scheduled at 07:30:00, but the schedule criterion was that it had to be scheduled within the window from 07:00:00 to 07:10:00. Figure 4 illustrates a sample rule from the ES knowledge base. The rule states that if a certain pair of Activity Planning Items are in conflict, then move the second relative to the first by a prescribed amount of time. The move is accomplished by deleting and then adding back the offending activity. The rule is generic and applies to many pairs of conflicts. Various instantiations of the rule appear in the packed rules dataset (two are shown in the figure: in the first instance API NNNN and API SSSS play the roles of API1 and API2 in Rule 36; in the second instance these roles are played by APIs V111 and W111). The prescribed amount of time that the second API must be moved to resolve the conflict is also provided in the dataset, corresponding to the particular instantiation involved. In the example the start of API2 (SSSS) must be moved to the start of API1 (NNNN) plus 7 minutes, and the stop 1 second after its start. Note that there are several additional fields available in the Packed Rules Dataset for the inclusion of additional APIs and scheduling constants, to allow for more complicated rules that may involve several APIs in their formulation.

FIGURE 3

AUTOMATIC ACTIVITY PLANNING

INPUT SYSTEM ACTIVITY PLAN

CFLT	*TIME*	*ACTIVITY*	*ID*	*START/STOP*
	08 FEB 06:50:00	A111	223	START
XX	08 FEB 07:00:00	NNNN	001	START
	08 FEB 07:12:41	R333	020	START
XX	08 FEB 07:30:00	SSSS	001	START
XX	08 FEB 07:30:01	SSSS	001	STOP

CONFLICT MESSAGES

```
08 FEB  07:00:00  SSSS  001  REQ  STRT  SCHED  WRONG  NNNN  07:30:00  07:00:00  07:10:00
```

Additional features of the Packed Rule Dataset include a priority field (PRI) and a branch code (BR). The priority field allows the user to direct the order in which conflicts are resolved; if a Key Activity is involved in a conflict and several Supporting Activities are in conflict as a result of that, the prioritization allows for the Key Activity conflict to be resolved first, relieving the need to resolve the concommitant supporting activities. The branch code is associated with the type of conflict; this allows modularization of the database so that not all rules need to be searched - only those with the branch code associated with the conflict type. Thus these two fields (PRI and BR) provide a way of efficiently chaining through only that subset of the rules that are of interest for that conflict.

Figure 5 illustrates the output of the ES. The new activity plan shows that API SSSS has been moved to start at 07:07:00, which is within the required window (07:00:00 to 07:10:00) relative to the start of API NNNN. There are no conflicts flagged, and a conflict resolution audit trail message restates the original conflict message and indicates the disposition. Incidentally, no new conflicts were introduced by Rule 36 because it was constructed by experts who knew how to resolve the conflict. However, to be doubly sure, the new activity plan is resubmitted to the conflict identification process used to flag conflicts in the first place. The combination of reconflicting and audit trail gives confidence to the user that the ES has done its job properly.

To summarize the key features of the ES:

design flexibility is achieved by use of packed rules together with the packed rules dataset which makes for an easily maintained and easily extended system;

speed is achieved through the use of branch code and priorities;

confidence is provided by the audit trail and by a final reconflicting.

The Interactive Graphics (IG) decision aid was designed to assist the operator to manually resolve those residual conflicts that were beyond the scope of the ES. It was rapidly prototyped in C on a Sun 3/110 workstation using the Sherrill-Lubinski graphics package. Recommended Human-Machine-Interface procedures were followed throughout. For example, all lines were doubly encoded: first with color, and second with line type (solid, dashed, dotted) to cater to the 10% of American men who are color-blind. Also, all colors are constructed by firing at least 15% of each color gun (Red, Blue, Green) for those operators who may be color-deficient. Early versions of the IG prototype were demonstrated to Activity Planning Operators, and their comments and suggestions were incorporated by fine-tuning the prototype. The IG fulfills the operator needs by providing static data base access (see Figure 2) and ease of use in editing.

FIGURE 4

AUTOMATIC SCHEDULING AND PLANNING

SAMPLE RULE

RULE 36

```
IF API1 IN CONFLICT = API1 IN PACKED RULES  AND
   API2 IN CONFLICT = API2 IN PACKED RULES

THEN

     DELETE API2 FROM PLAN

     COMPUTE NEW START/STOP TIMES FOR API2 BIASED
     FROM API1

          API2 START = API1 START + DATABASE VALUE 1
          API2 STOP  = API2 START + DATABASE VALUE 2

     ADD API2 INTO PLAN AT TIME COMPUTED ABOVE

ENDIF
```

PACKED RULES DATASET

PRI	BR	RULE	API IDENTIFIERS						SCHEDULING CONSTANTS					
1	4	13	Q111	R111	XXXX	XXXX	XXXX	XXXX	1000	700	0	0	0	0
2	6	21	T111	U111	XXXX	XXXX	XXXX	XXXX	300	1500	90	105	0	0
1	5	36	SSSS	NNNN	XXXX	XXXX	XXXX	XXXX	7	1	0	0	0	0
1	5	36	V111	W111	XXXX	XXXX	XXXX	XXXX	300	1500	0	0	0	0

FIGURE 5

AUTOMATIC SCHEDULING AND PLANNING

OUTPUT SYSTEM ACTIVITY PLAN

CFLT	TIME	ACTIVITY	ID	START/STOP
	08 FEB 06:50:00	A111	223	START
	08 FEB 07:00:00	NNNN	001	START
	08 FEB 07:07:00	SSSS	001	START
	08 FEB 07:07:01	SSSS	001	STOP
	08 FEB 07:12:41	R333	020	START

RESOLUTION AUDIT TRAIL

```
08 FEB  07:00:00  NNNN  001  REQ  STRT  SCHED  WRONG  SSSS  07:30:00 07:00:00  07:10:00
CONFLICT RESOLVED.  RULE NUMBER 36 USED.  SSSS MOVED TO NNNN START.
```

CONCLUSIONS

- GE's Activity Planning procedure in which all requests for activities are accepted, merged, sorted by start times and then checked for conflicts using schedule criteria, conflict criteria and resource limits was found to be the most appropriate for the unique problems faced; no other scheme in the literature appeared better.

- It is feasible to expect to resolve most of the conflicts automatically by a simple Expert System

- The Expert System rules require one to four hours each to knowledge engineer, but hundreds rather than thousands of rules are probably adequate.

- The use of 'packed rules' together with a 'packed rule dataset' makes for a highly efficient, easily maintainable implementation.

- Some conflicts require manual intervention; Interactive Graphics can be a valuable aid to the operator.

- Techniques to speed up the Expert System execution time include use of a branch code to segment the rule base and use of rule priorities to eliminate unnecessary resolution of Support Activity conflicts.

ACKNOWLEDGMENTS

A great debt of gratitude is acknowledged for the superlative contributions of Sam Davis, who helped design, knowledge-engineered and programmed the Expert System, and helped design and programmed the Interactive Graphics prototype. Thanks also to Peggy Frederick and Mike Baluta, our planning experts.

N89 - 15604

A Dynamic Case-Based Planning System for Space Station Application*

F. Oppacher
D. Deugo

School of Computer Science,
Carleton University, Ottawa, Ontario,
Canada K1S 5B6.

Abstract

We are currently investigating the use of a case-based reasoning approach to develop a dynamic planning system. The dynamic planning system - DPS for short - is designed to perform resource management, i.e. to efficiently schedule tasks both with and without failed components. Our approach deviates from related work on scheduling and on planning in AI in several respects. In particular, we attempt to equip the planner with an ability to cope with a changing environment by dynamic replanning, to handle resource constraints and feedback, and to achieve some robustness and autonomy through plan learning by dynamic memory techniques. We briefly describe the proposed architecture of DPS and its four major components: the PLANNER, the plan EXECUTOR, the dynamic REPLANNER, and the plan EVALUATOR. The planner, which is implemented in Smalltalk, is being evaluated for use in connection with the Space Station Mobile Service System (MSS).

1. Introduction

In real-world planning tasks it is often necessary to manage plans that contribute to more than one goal, to flexibly adjust plans that conflict with concurrent goals, and to anticipate and avoid bad planning. Moreover, to achieve some degree of autonomy, a planning system that may have to operate in changing environments must rely on feedback. The presence of feedback presupposes an ability for dynamic replanning, i.e. for reacting to changing conditions as execution proceeds. Feedback tasks also often involve tight time and other resource constraints.

We have argued in [Deugo et al. 88] that the feedback-imposed needs for dynamic replanning and for dealing with continuous resources like power or time are beyond the capabilities of traditional schedulers such as, e.g., PERT and CPM [Moder and Phillips 70], and AI planners such as, e.g., STRIPS [Nilsson 80] and NOAH [Rich 83]. For example, a STRIPS-like approach seems to presuppose that the system's world model is and remains correct, that the operator always does exactly what is required, that nothing happens between making the plan and executing it, and that the plan is executed precisely as planned. In short, such an approach works only in static situations and not in more realistic settings. Postponed commitment planners [Stefik 81] attempt to solve this problem by postponing the commitment of the exact order in which their task are to be executed until execution time. However, the size of the partially ordered set of tasks constructed can be exponential in size to the set of tasks, and may not account for every type of situation.

---

* This research is undertaken on behalf of the Department of Communication, Communications Research Centre, with funding provided by the Canadian Space Station Program Office of the National Research Council.

Realistic planners should also be able to adapt old plans to the current situation and to extend their plan library by learning. A robust and efficient planner should neither be forced to give up if there is no appropriate, ready-made plan in its library nor have to replan always from scratch.

We believe that a combination of dynamic memory techniques [Kolodner 84] [Schank 82] and case-based reasoning techniques [Hammond 86] [Kolodner et al. 85] is necessary to successfully tackle all of these problems. Section 2 outlines how the architecture of the DPS integrates several dynamic memory and case-based techniques and identifies the relationships among its major components. Section 3 describes these components in more detail, and section 4 proposes extensions to our approach and summarizes.

2. DPS Architecture

The tasks to be planned for by the DPS depend on the availability of different resources such as time and power. A human operator enters information about resource availability in the form of initial plan constraints. The DPS relies on a feedback loop to provide information about the success or failure of the plan's execution. Subsequent planning sessions involving this or a similar plan can use this information and thereby gain from past experience. Thus, starting with an initial library of plans, the DPS acquires new plans through a form of plan learning from past plans.

Our dynamic memory and case-based approach to the planning problem postulates four major components: the PLANNER, the plan EXECUTOR, the dynamic REPLANNER, and the plan EVALUATOR. Figure 1 shows how these components fit together.

The PLANNER controls the planning process, from information input, past plan locating, to plan construction. Initially the operator configures the planner with its resource information. This provides the resources the planner can use over the plan execution period. Next, the operator enters the planning parameters, i.e. the tasks and constraints, to help set up the plan construction phase. The Locator uses a case-based approach to locate a past similar plan-goal-resources configuration. Using the supplied input information, the Locator indexes into a library of old plans, indexed by their goals (tasks), in an attempt to find a plan that matches the current planning parameters. If a matching plan can be found, it is passed to the plan EXECUTOR component. If no plan can be found, the Locator attempts to find a plan whose goals are a subset or superset or generalizations or specializations of the current goals. This past plan is then modified by the Constructor's domain planning information or planning heuristics found in the knowledge base, or by the operator, to create a new plan to be executed by the plan EXECUTOR component. The index to the knowledge about planning and plan modification is formed using the task and constraint information about the task the planner is currently considering. The modified plan is then verified using expected resource information to insure the plan's integrity. A failure in a task's verification will cause the Constructor to alter the plan.

This approach enables the planner to continue planning even though it has no exact ready-made plan to deal with the current tasks.

The Explainer collects all the planning activity information and can then explain to the operator what the plan is and how it was constructed. In addition to this, it can also use past plan exception information to describe failure conditions that could arise in the execution of the plan. This information is also used by the Constructor to help build better plans, or by the operator to help alter the Constructor's proposed plan. Figure 2 provides an example of the planning activity information stored by the Constructor when an existing plan is modified to meet the current goals.

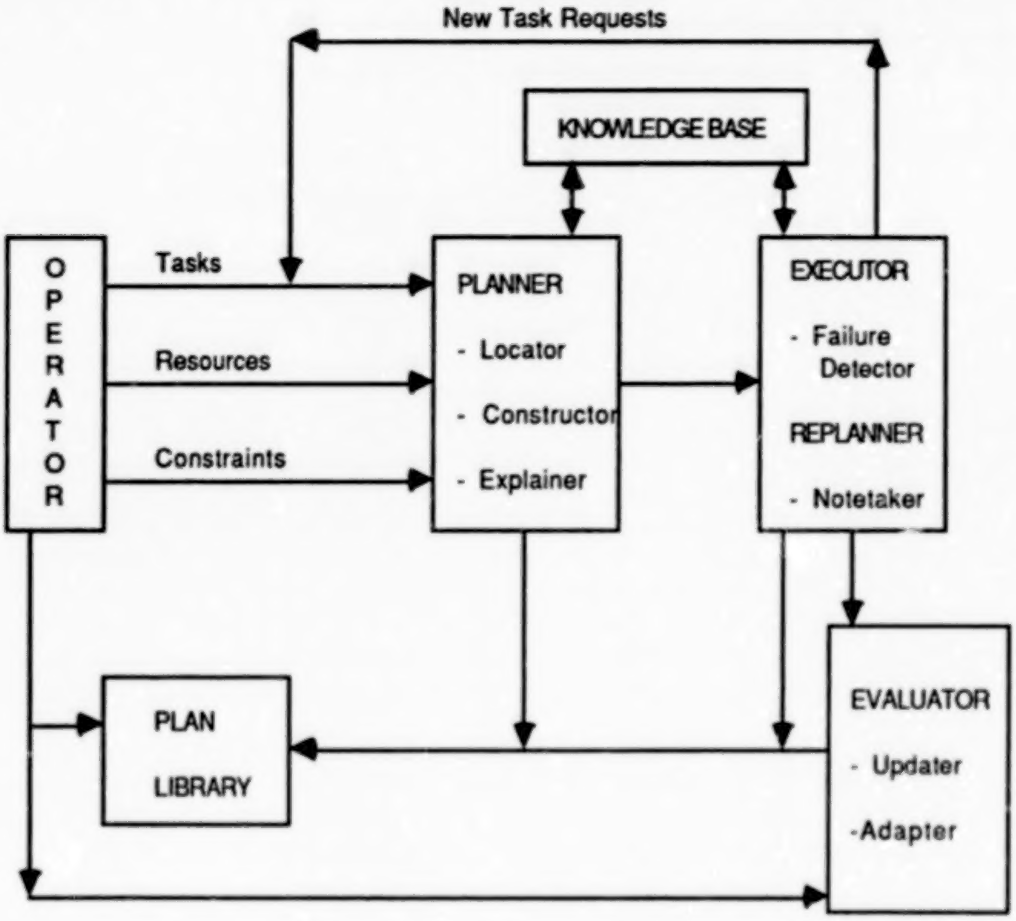


Figure 1. Architecture of the DPS

The plan EXECUTOR component takes the plan and starts the execution of it. If the Failure Detector experiences a failure condition (lack of resource) that was unforeseeable at the time of plan construction and arose during the plan's execution, the exception is noted by the Notetaker, and the dynamic REPLANNER component is activated. This component will attempt to reorder the plan, remove failing goals, or ask the operator for assistance in order to keep the EXECUTOR's plan execution continuing. These actions are found in the knowledge base and are indexed like any other planning information. After all, when a plan fails one does not want to stop the execution since resources have been allocated and are ready to use. The information about what replanning was done is noted by the Notetaker for later use by the EVALUATOR component.

The dynamic REPLANNER component is an important improvement that distinguishes our planner from other planners described in the literature. It prevents minor faults from stopping plan execution, and causes only moderate modifications of the plan. The REPLANNER uses the planning technique known as goal planning or reactive planning [Schoppers 87] to keep a plan executing. It also indexes into old plans to see if any replanning information is available for use in the current situation.

[1] Plan B selected with a rating of one extra task, one missing task, and no failures.
[2] Adding missing tasks, to Plan B.
[3] Removing extra tasks, from Plan B.
[4] Verifying Plan B.
[5] Inspection task failed, replanning.
[6] Rule 'Move task to end of Plan' fired, executing action rule.
[7] Verifying Plan B.
[8] Plan verified, ready for execution.

Figure 2. Explainer's Planning Activity Information

After the plan has executed, the plan and the information provided by the Notetaker, i.e. exceptions and replanning descriptions, are given to the EVALUATOR component. If the plan was an old plan that executed successfully, this information is added to the plan in the plan library to provide added support information for it. If the plan was newly created and it executed successfully, it is added to the plan library along with the goals it satisfied. If the plan failed, the exceptions and replanning information are recorded in the plan along with the reasons why the goal, or goals, failed. All of these transactions are handled by the Updater. If the plan has had a bad track record, it may also be altered by the Updater using the Notetaker information to make it a 'better plan' in the future. An updated plan is 'better' than the original plan because either tasks with a proven history of failure have been removed from it or it includes Notetaker information which can be used in future planning sessions. Failure and success information are valuable in determining the best plan for the current situation. The operator is also part of this activity: he/she helps to verify the reasoning of the Updater, and ensures the sanity of updates for the plan library.

The EVALUATOR component helps the planner acquire new plans and knowledge by learning from itself. This is achieved by adding new, successfully executed plans constructed by the Constructor, and by altering old plans due to planning failures. By recording the failures, the EVALUATOR also learns to fix and adapt plans to new environments over time.

The PLANNER, REPLANNER, and EVALUATOR components rely on dynamic memory and case-based techniques to generate a plan, to alter a plan due to a failure, to store new plans, and to update old ones. These techniques enable the DPS to work in a dynamic environment and be ready to meet a wide range of unforeseen changes.

3. Component Definitions

The inputs to PLANNER consist of the plan resources, the tasks to be planned, and the constraints on the tasks. In our prototype implementation this information is provided by a human operator. The output of the PLANNER is a plan which is later executed by the EXECUTOR and updated or added to the plan library. We now briefly discuss each part.

Resources can take the form of any type of (usually time-varying) physical supply, such as electrical power. In some cases, the consumption of a resource can increase or decrease the supply of another. This is know as a Supplied resource. The planner must keep track of all available resources. Resources are defined by resource functions that enable the planner to predict the supplies at time t.

Tasks are treated by the current implementation as unit activities that cannot be further decomposed. The DPS schedules tasks but does not plan for the execution of individual tasks. They are assumed to be directly executable by the EXECUTOR; future extensions will allow the

PLANNER to be applied to tasks as well, thereby facilitating the construction of hierarchical plans. A task's information aids the planner to efficiently position it among the other tasks in the plan. This information is entered by the operator using a form-based approach. A task form with data slots identifying the task, its constraints, its requirements, and the supply of different resources it increases, is provided for the operator to fill in. Thus, a task has the following structure:

```
(task-name-slot                ; the name of the task; e.g., operation X.
activity-slot                  ; what the task is to do; e.g., running operation X.
constraint-1-slot              ; e.g., must be done by 0800 hours.
    •
    •
constraint-n-slot
resource-required-1-slot       ; e.g., operation X uses 50 watt hours of electricity.

    •
    •
resource-required-n-slot
resource-supplied-1-slot       ; e.g., operation X raises the temperature by 1°C.
    •
    •
resource-supplied-n-slot).
```

Constraints are identical to task constraints, except that they constrain a plan. For example, a plan constraint could be that the plan must start execution before 0800 hours and finish execution by 0900 hours. The operator supplies this information to the planner by entering the data on a plan constraint form.

A Plan is an ordered sequence of tasks, produced by the PLANNER using the initial planning information, for execution by the EXECUTOR. Each task in a plan has two new slots added to it, a start-time-slot and finish-time-slot, which are used by the EXECUTOR to determine when each task is to start and finish execution. A plan has the following structure:

```
(plan-name-slot                ; the name of the plan.
success-slot                   ; a count of the number of times, initially zero, the plan
                                 has executed successfully without having to be
                                 replanned by the Replanner.
failure-1-slot                 ; failure slots include information about how the plan
                                 failed and what was done to correct it.

failure-2-slot
    "
    "
failure-n-slot
task-1-slot                    ; points to component task; e.g., operation X.
task-2-slot
    "
    "
task-n-slot).
```

The **Plan Library**, as the name suggests, is a library of past plans that have either been created initially by the operator and entered into the library, or have been created by the system and added to the library. They are stored sequentially and are indexed by the search mechanisms of the Constructor component of the PLANNER. The library is memory-resident in the current implementation but will be converted to a database management system when issues such as size, access, and information updating become important.

The **PLANNER** develops a plan to execute the operator-entered tasks. It uses past plans, stored in the plan library, in its attempt to locate or build a new plan. The PLANNER consists of three components: the Locator, the Constructor, and the Explainer.

The **Locator** takes all of the tasks' activity-slot identifiers and tries to locate a past plan that contains only those tasks. If such a plan is found in the plan library, it is passed to the Constructor. If no plan can be found, the Locator looks for a past plan whose tasks properly include the current requirements. If such a plan can be found, it is passed to the Constructor with the tasks in excess of the current requirements marked. If still no plan can be found, the Locator looks for a past plan that contains the maximum subset of tasks currently required. This plan is passed to the Constructor with the tasks missing from the plan identified. When multiple plans are located, the one with the best success rate, calculated by subtracting its successes from its failures, is returned.

The **Constructor** first checks to see if the plan contains only the required tasks. Any excess tasks are simply removed. If it has less, the Constructor rules stored in the knowledge base are accessed to decide what action to take. Specific Constructor rules have both a task and a constraint as rule identifiers, and are considered first. If there are no such rules, more general rules are accessed that rely only on the task or the constraints, but not both. The actions taken by such rules include: to append a task to the end of the plan, to put it at the front of the plan, to put it after a specific task in the plan, or to find the first available position that satisfies its constraints.

Using the resource function information, the plan is verified to ensure that all task and plan constraints are met. If a constraint fails, the combination of constraint failure and task is used as an index to a rule which provides the appropriate action to be taken with the plan. Such actions could take the form of: removing the task, delaying it until its resource requirements are met, or asking the operator for help. These constraint rules depend on the type of constraints and tasks handled. Once the plan is constructed, it can be verified or altered by the operator if desired, and then passed to the Executor.

The **Explainer** component describes what actions were taken to create the plan. It identifies what and why past plans were chosen, the problems and successes the past plan had, what actions were done in creating the current plan from the past plan, and what constraint problems were found and solved for the plan. The Explainer can be turned on during plan generation to allow the operator to view the creation process of the plan. Alternatively, the operator can ask individual questions at the end of the planning phase.

The **EXECUTOR**, using the planning information, executes the plan. It sequentially takes each task in the plan and performs that task's activity. A task finishes when its task activity ends or when its finish-time is reached. Information about the execution of the plan is stored by the Notetaker. At plan completion, the plan and its execution information are passed to the EVALUATOR.

For implementation purposes, a plan executes in discrete time slices. At each time slice, the operator can view the current state of the plan, the task executing, and the resource information. He can also vary the resource information in this period to cause a resource failure of the task, thereby forcing the replanning mechanism.

The **Failure Detector** monitors the resource sensors to ensure that none of a task's constraints are being violated. If at any time a task constraint is violated, the violation is noted by the Notetaker, the plan execution is stopped, and control is passed to the REPLANNER. The

REPLANNER will produce an adapted plan and return it to the EXECUTOR to restart execution from the point of interruption.

The **REPLANNER** first checks to see whether a given failure has been detected before. This is done by looking in the past plan on which the current plan is based. If it had failed in a similar manner before, the replanning information stored in the past plan is used to adapt the current plan, and the newly adapted plan is passed back to the EXECUTOR to commence execution. If there is no replanning information, the REPLANNER consults its replanning knowledge base. It uses the task and failing constraint to index a replanning rule to adapt the current plan. These rules are a subset of the Constructor rules and have the same form. Their actions may consist of deleting the task, delaying it, repositioning it, stopping the plan's execution altogether, or asking the operator for help. The actions taken by the REPLANNER are recorded by the Notetaker before the adapted plan is passed back to the EXECUTOR.

The **Notetaker** is responsible for recording all information about the execution of a plan, in particular its successes and failures. In the case of failures, the Notetaker records the failure causes and any replanning information. This includes information about successful replanning episodes and the replanning rules used by them. Thus, the Notetaker fills in the following information slots:

Success-slot - Initially true. If a plan fails, it is set to false.
Number of failures - Initially zero, incremented by one for each failure.
Failure-slot - Initially empty. One slot is created and filled in per failure type. A task's failure-slot identifies the task executing, the failure-type, the failure-cause, the replan-type, and the failure-count.

The **EVALUATOR** receives the plan and the Notetaker information about it, and must decide what actions should be taken with the plan. It has several options depending on the plan type and the success or failure information. These options include:

Success of Old Plan - If the plan was previously used, its success-slot is incremented to boost its strength.
Success of New Plan - If the plan is a new one and it executed successfully, it will be added to the plan library with its success-slot set to one.
Failure of New Plan - If the plan had a minor failure (for example, one task out of fifty was delayed), the plan may be added to the library with a failure-slot filled in. If the failure has occurred often, the plan is discarded as being invalid.
Failure of Old Plan - If a task in a plan failed for the first time, the failure information is added to a new failure-slot for that task in the plan library. If a failure of this type already exists in a task's failure-slot, the failure-slot-count is incremented.

Just as the Constructor 'massages' the retrieved plan's constraints and tasks to make them fit the current situation, so too does the EVALUATOR. Using knowledge base rules, it massages the failure conditions so that they are adapted and appropriate for the current plan and its situation.

The **Updater** has the job of updating or adding a new plan or plan information to the plan library. Its algorithm is based on the four possible types of updates identified by the EVALUATOR. Updates can be monitored and, if it is desired to maintain close control over the reasonableness of the evolving plan library, modified by the operator.

The **Adapter** reorganizes plans that have failed often to improve them for future use. It uses replanning information in the failure-slots to determine whether to remove a task from the plan, delay the task in the plan, reorder the task, or remove the plan from the library. The Adapter will be activated when plan failures have reached a preset threshold.

Another part of its task is to look for several plans that can be generalized into a single abstract plan that preserves all of the information of the other plans. The latter can then be removed from the library. This reduction of the number of plans improves the search efficiency without compromising the planning knowledge in the library. It also helps prevent the library from filling up with many similar plans and provides plans that can be used in many different situations. We intend eventually to apply this process of generalization in the context of problem solving to the rules used by the Constructor and Replanner, thus keeping redundant rules from cluttering the knowledge base.

4. Summary

As can be seen from this brief description, our planner is concerned with many of the different areas of case-based reasoning. We retrieve past cases based on the number and type of tasks in the current situation they match, and use the number of times a plan succeeds in similar situations. We use a knowledge base of rules to aid in the plan transformation of a past plan to the current situation. We use past planning information and current planning operations to explain the planning task. We do dynamic replanning using the same knowledge base to keep the plan executing. We note all of this information and encode it into a past plan or new plan, enabling plan cases to be better utilized on the next planning iteration. The feedback loop enables plans and new plan learning to evolve with the environment over time.

By using dynamic memory and case-based reasoning techniques, combined with knowledge based techniques for replanning, we have presented a design that handles resource constraints, feedback, and achieves both robustness and some degree of autonomy through plan learning. Although not essential to any part, the operator can guide the overall planning process and control the acquisition of new plans and rules for replanning. With the current design and the future enhancements, we feel we are approaching a realistic, efficient planner.

Planned enhancements to the DPS not already mentioned include improving the process of unifying the supplied operator planning information to that of a stored plan, unifying all of the planning information, not just the goals, and when the supplied information is incomplete. Also, the DPS will be enhanced to recognize dangers and opportunities during the plan's execution. This will allow the plan to benefit from or avoid problems during execution in its current environment. A final enhancement will be to take the plan's failure information and generalize it into an action recommendation. This recommendation is used in replanning before the specific task/constraint failure information is accessed. By doing this, additional planning failures should be avoided after replanning because the recommendation has already taken them into account.

5. Acknowledgments

We wish to thank Prof. D. Thomas of the School of Computer Science, Carleton University, P.J. Adamovits and R.A. Millar of the Department of Communication, Communications Research Centre, and the the Canadian Space Station Project Office for supporting this work.

6. References

Deugo, D. L., Oppacher, F., Thomas, D., Planning Techniques Survey: Their Applicability to the Mobile Service System, TR/SCS, Carleton University, Ottawa, 1988.

Hammond, K. J., CHEF: A Model of Case-based Planning, AAAI 1986, pp. 267-271.

Kolodner, J. L., Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model, Lawrence Erlbaum Associates, Publishers, 1984.

Kolodner, J. L., Simpson, R. L., Sycara-Cyranski K. L., A Process Model of Case-Based Reasoning in Problem Solving, IJCAI 1985, pp. 284-290.

Moder, J. J., Phillips, C. R., Project Management with CPM and PERT, Van Nostrand Reinhold Company, 1970.

Nilsson, Nils J., Principles of Artificial Intelligence, Palo Alto, California Tioga Press, 1980, pp. 275-360.

Rich, E., Artificial Intelligence, McGraw-Hill Book Company, 1983, pp. 247-294.

Schank, R. C., Dynamic Memory: A Theory of Reminding and Learning in Computers and People, Cambridge University Press, 1982.

Schoppers, M. J., Universal Plans for Reactive Robots, IJCA 1987, 1039-1046.

Stefik, M., Planning with Constraints (MOLGEN: Part 1), Artificial Intelligence, 16, 2, May 1981, pp. 111-140.

N89-15605

TOWARDS A KNOWLEDGE-BASED SYSTEM TO ASSIST THE BRAZILIAN DATA-COLLECTING SYSTEM OPERATION

RODRIGUES, V.; SIMONI, P.O.; OLIVEIRA, P.P.B.;
OLIVEIRA, C.A.; NOGUEIRA, C.A.M.

Institute for Space Research
Computing and Applied Mathematics Laboratory
Artificial Intelligence Group

Caixa Postal 515
12201 - São José dos Campos - SP
Brazil

ABSTRACT

In this paper it is reported a study carried out to show how a knowledge-based approach would lead to a flexible tool to assist the operation task in a satellite-based environmental data collection. Some characteristics of a hypothesized system comprised of a satellite and a network of Interrogable Data Collecting Platforms (IDCPs) are pointed out. It is briefly described the Knowledge-Based Planning Assistant System (KBPAS) and some aspects about how knowledge is organized in the IDCP's domain.

1 - INTRODUCTION

The first phase of the Brazilian Complete Space Mission (MECB) is to be accomplished with the launching of the first Brazilian Data Collecting Satellite (SCD1). The system will offer capabilities for satellite-based environmental data collection.

This system will comprise: a) a network of fixed "Data Collecting Platforms (DCPs)" deployed at the Brazilian territory both on land and in the air, all of them independently transmitting the environmental data. The DCPs will be equipped with only one transmitter for uplinking messages to the satellite; b) the Satellite, which will have the following main characteristics: low circular orbit around Earth with average altitude about 750 km and nominal inclination about 25 degrees with relation to the Earth equatorial plane. It will retransmit to a ground-station the environmental data received from the DCPs, and it will receive the DCP messages on a random access basis; c) the Ground Station and the Processing Center, where data will be received, processed and distributed to users.

PRECEDING PAGE BLANK NOT FILMED

The SCD1 will be able to receive message transmitted only by platforms within its visibility area and to retransmit these messages when the satellite will be over the ground station visibility area. In addition, the access to the satellite by some DCPs will also be dependent on the number of platforms under its sight, the satellite-platform relative position, the platform repetition period, the message length, the transmission power and other secondary factors [1].

Considering that due to a design option all DCPs will transmit in the same frequency, the received frequencies by the satellite will be randomly distributed because of the difference in Doppler shifts associated with a random geographic distribution of DCPs on the Brazilian territory. Therefore the main predicted drawback will the risk of non-acquisition of messages from some DCP, due either to interferences from simultaneous transmission (which may produce the same received frequency), or to the unavailability of a transponder unit, since the Satellite will have only two transponder units [1]. In fact, during the interval when the satellite is retransmitting a specific DCP data all other DCP message transmissions would not be received and consequently their data would be lost; in order to avoid this problem the DCPs will repeat their messages several times during a satellite overpass.

The system ought to operate all the time, attempting to obtain maximum availability and minimum data loss. However,it has been difficult to know "a priori" what will be the "system use factor" defined as the message mean arrival rate at the receiver. Taking into account simulation studies about a similar system, the ARGOS system [2], it can be estimated that the satellite will be able to handle up to 1000 DCPs simultaneously with message duration randomly distributed between 0.36 and 0.925 seconds.

From the preceding observations, it is clear that the system would not be flexible enough to select the most important DCPs at each pass, even though their high message repetition rate may enhance their likelihood of accessing the Satellite.

Considering this problem, it has been carried out a study about a hypothesized system composed of: a) a network of "Interrogable Data Collecting Platforms (IDCPs)". It means that the satellite will have the ability to select, in a particular time, a particular IDCP. The DCPs in this system will be more complex, because they should be equipped with a receiver and a transmitter; b) a satellite with the capability of interrogating an IDCP and also of receiving and retransmitting its data immediately afterwards; c) the processing center with the capability of planning which platforms will be interrogated before each satellite pass.In this case the network operation will need an expert operator that should integrate every necessary information to attain efficient and flexible network utilization.

In this paper some issues concerned with a knowledge-based approach to assist the IDCPs network operation are presented and discussed. Some problems concerning the use of knowledge-based technology are presented and it is also proposed a model for the Knowledge-Based Planning Assistant System (KBPAS).

2 - USING KNOWLEDGE-BASED TECHNOLOGY

The knowledge-based approach may play an important role in the development of advanced IDCPs network operation systems, where the integration of multiple and disparate sources of information is needed. Some interesting comments about knowledge-based systems may be found in [3].

2.1 - KNOWLEDGE USE IN THE IDCPs NETWORK OPERATION TASK

In an initial analysis it is suggested that if the IDCPs network is to operate efficiently, it is plausible to assume that expertise will be developed in order to attain near optimal net management. It means that the net and its operation should have an evolutionary design, in which the initial configuration would require a manual interaction (it is expected that an expertise will be created). As the number of IDCPs increase it will be required a computational system to assist the operation, generating an adequate plan (a list of most important IDCPs for the next passage).

For these reasons, all required knowledge will be taken as a "plausible" one, whose initial representation and utilization would be guidelines for the expertise to be acquired subsequently.

2.2 - RESPONSE TIME

An observed drawback of many existing knowledge-based systems is their slow response time. An assistant system for IDCPs operation task has not the requirement of quick response, although this kind of system admits a real-time conception.

The communication between the Ground-Station and the Satellite (to transmit the plan for the next interrogations) will occur when the satellite enters the Ground-Station visibility area. The maximum system response time is the time interval between two passes, which represents a safety margin for the worst case, i.e., the situation in which some information concerned with the last pass will also be taken into account for the next passage plan.

C-6

2.3 - EXPLANATION CAPABILITY

This explanation capability should support both the system development and operational phases. In the former it will assist the knowledge base debugging, while in the operational phase it would provide facilities to explain:

* why a given IDCP was not included in the last generated plan;

* which were the most important factors that influenced a certain IDCP selection for a particular passage;

* which were the conflicting factors that influenced a certain IDCP selection for a particular passage.

In addition, the system will present the ability to exhibit a detailed trace of the reasoning chains, which is a desirable feature for an assistant system [4]. All these explanation capabilities will result in reports.

It is intended that the system will operate autonomously when the network will be in its full operational status, although the explanation capability will always be available.

3 - THE KNOWLEDGE-BASED PLANNING ASSISTANT SYSTEM (KBPAS)

To guide the KBPAS design efforts, it may be guessed a scenario that exemplifies a typical and non-trivial IDCPs interrogation planning problem. This scenario assumes a non-uniform IDCPs density distribution across the Brazilian territory. In a first phase only the Brazilian territory is considered; however if it is assumed some satellite buffering and storing capability other countries could also use these facilities. This scenario is intended to allow the representation and management of typical seasonal meteorological occurrences, abnormal environmental phenomena, failures (partial or total), cloud coverage, transmission power attenuation problems and other circumstancial factors.

A simplified model of the KBPAS which is the basis for this study is shown in Figure 1. Its overall structure comprises five major components: the user interface, the IDCPs past plans base, the high and low level planners, the knowledge base and the database.

The user interface should be designed to easily display IDCP network operation information, such as the IDCPs and satellite geographic positions, the status and graphs of relevant parameters, and warning indicators.

The user may have access to reports showing either IDCPs proposed plans in past passages or the plan inferred for the next pass, permitting in this way "a posteriori" plan analysis; in addition, the user would be allowed to alter directly the plans at any of the two planning levels.

The knowledge base and reasoning: considering the complexity of the information upon which operation techniques are based, an organizational framework for this information is recommended, in which it will be integrated taking into account the uncertainty and the need of an indexing structure.

Basically, the IDCPs will be distributed across regions in the states (or even sub-regions) as needed. The scattering of the IDCPs at the Brazilian territory will entail a non-uniform density, and at each pass the satellite visibility area will cover partially this IDCPs network. Besides this available information in the database, there will also be the message length, the repetition period, etc.

The IDCPs in the network may be requested by its features, such as the particular area where they are placed, their label as they are related to their applications (hydrological, meteorological or snow data stations), etc.

The types of requisition available are related to:

* abnormal environmental phenomena monitoring (forest fire, drought, flood, hoar-frost or snow, pollution, intense hot weather, etc);

* normal experimental needs (e.g., meteorological studies);

* particular needs (e.g., strategic);

* the impossibility of data collecting in past passages (partial or total IDCP failures, satellite temporary problems);

* global and local atmospheric research programs.

It is envisaged a priority scheme that may rule out conflicts between competing requisitions.

Each plan to be generated is concerned with the next satellite pass. The high level planner should reason in a data-driven way generating a list of IDCPs that should be interrogated along the next passage, one station at a time. An embedded feature of the program that generates the plans is to take into account the information about which IDCPs were accessed and did not answer as expected.

That list of IDCPs is then passed to the low level planner that must order it on adequate time intervals by incorporating the information of satellite position along its orbit, and transform it in an effective sequence of telecommands that should be transmitted to the satellite.

The already executed plans may be stored in the past plans base by specifying the date, the orbit data and the high and low level plan representations. This information may permit some plan statistics or the analysis of a particular plan.

The initial IDCPs information to be integrated is shown in Table 1.

4 - CONCLUDING REMARKS

As it was pointed out earlier, this study was aimed at investigating the feasibility of a knowledge-based system to assist the IDCPs network operation. It is believed that the use of knowledge-based techniques may greatly expand the capabilities of the satellite-based environmental data collecting system, and can give flexibility in the evolutionary aspect of the IDCPs network operation system development.

REFERENCES

[1] ROSENFELD, P. MECB Ground segment: systems aspects. A-REV-0030, 1987

[2] LOCATION AND DATA COLLECTION SATELLITE SYSTEM USER'S GUIDE SERVICE ARGOS

[3] WATERMAN, D. A. A guide to expert system. Reading, K.A., Addison-Wesley, 1986

[4] BUCHANAN, B. G. ; SHORTLIFE, E. H. Rule-based expert systems. Addison-Wesley, Reading, Mass. 1985

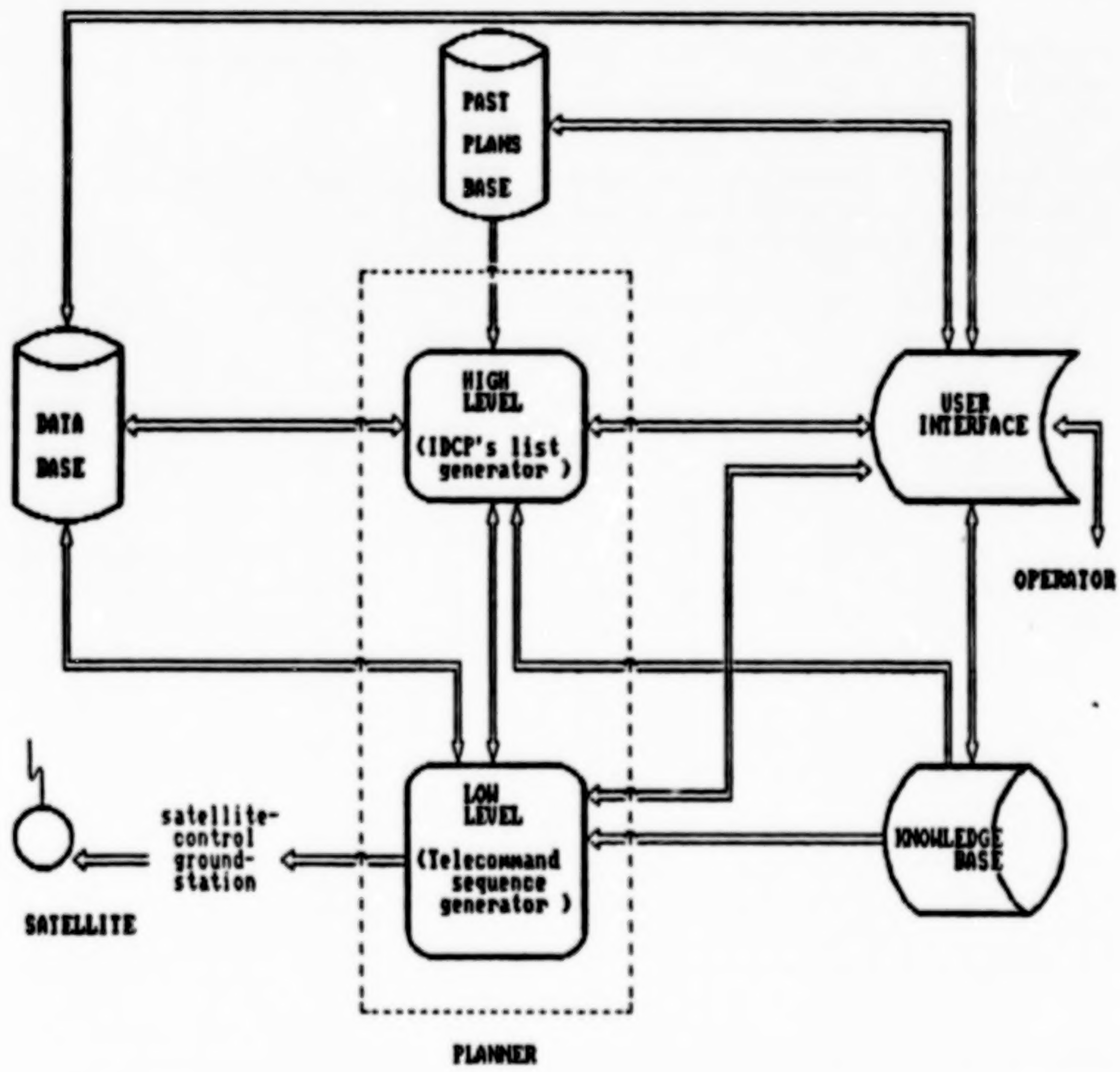


Figure 1 - The KBPAS architecture and ground-station facilities.

TABLE 1 - THE IDCP PARAMETER INFORMATION

Parameters	Value
1. Cloud coverage expected at the IDCP position for the next satellite passage.	cover/clear
2. The IDCP position in the satellite visibility area for the next passage.	within/out
3. The IDCP position in the satellite visibility area for the second next passage	within/out
4. Indicator if IDCP data was received during the last satellite pass.	yes/no
5. The environmental phenomenon type at the DCP location .	normal/abnormal
6. The kind of normal environmental phenomenon at the DCP location.	one of the possible meteorological/ climatological phenomena (rain, snow, etc)
7. The priority assigned to each IDCP (three priority degrees).	low/medium/high
8. The IDCPs spatial density in its surrounding (for example: number of IDCPs present in a limited area).	numerical
9. The IDCP operational status.	normal/partial failure/total failure
10. The IDCP operation life time after its installation (it suggests failure possibility and power transmission level attenuattion).	short/long

Parameters	Value
11. The expected IDCP repetition period. It is the time interval between two consecutive messages. It must be chosen in order to allow a sufficient data sampling.	numerical
12. The IDCP message duration. It is the sensor data string duration time. It depends on the number of sensors connected to the IDCP.	numerical
13. The IDCP last reception date (the time interval between this date and the next pass is a plausible indicator of the environmental data sampling rate).	numerical
14. The sucessful IDCPs reception rate (it is compared with the expected sampling rate).	numerical
15. The specific demand for receiving data from the IDCP.	yes/no

N89-15606

Use of an Expert System Data Analysis Manager for Space Shuttle Main Engine Test Evaluation

Ken Abernethy
Furman University
Greenville, SC 29613

Abstract

The ability to articulate, collect, and automate the application of the expertise needed for the analysis of space shuttle main engine (SSME) test data would be of great benefit to NASA liquid rocket engine experts. This paper describes a project whose goal is to build a rule-based expert system which incorporates such expertise. Experiential expertise, collected directly from the experts currently involved in SSME data analysis, is used to build a rule base to identify engine anomalies similar to those analyzed previously. Additionally, an alternate method of expertise capture is being explored. This method would generate rules inductively based on calculations made using a theoretical model of the SSME's operation. The latter rules would be capable of diagnosing anomalies which may not have appeared before, but whose effects can be predicted by the theoretical model.

Introduction

The analysis and interpretation of SSME test data presents some significant challenges. A single SSME test can produce in excess of 50 megabytes of data, with current test schedules calling for more than 10 tests per month. In addition, the complexity of the SSME reduces the possibility of such data analysis becoming routine and creates a requirement for high levels of data analysis expertise. As a consequence of these factors, there are rarely enough experts or time for a completely optimal SSME test data analysis.

A large portion of the expertise being used in such analysis is in the form of experiential knowledge gathered and possessed by engineers with years of experience in the task of analyzing SSME test data. Much of this expertise is unavailable to novice engineers, because it has not been articulated and recorded in accessible forms. Thus the possibility of losing significant expertise and analysis capability through the loss of personnel is even higher than in many other expert dependent problem domains. Further, it would be advantageous for NASA to increase the use of its most experienced liquid rocket engine experts in high-level planning and design activities. For this to become possible, some of the burden of SSME data analysis and interpretation must be passed to junior engineers and/or automated.

For the reasons outlined above, engineers in the Propulsion Systems Division at Marshall Space Flight Center recognize that it is highly desirable to articulate and capture SSME data analysis expertise and to automate, or at least reduce the level of expertise required for, some components of the analysis of SSME test data. The

PRECEDING PAGE BLANK NOT FILMED

project described in this paper is an attempt to move toward these goals through the construction of an SSME expert system data analysis manager.

An effort has been underway for the past several years at Rocketdyne (NASA's primary contractor for the SSME) to incorporate expert system technology into SSME test data analysis, c.f. [2], [3], [4], and [6]. The approach taken in that effort has been to use a selected database of SSME test results as source information for the inductive generation of expert system rules. The inductive-based expert system building tool ExTran7 has been the primary tool used.

The current effort at Marshall Space Flight Center adopts an approach different from that taken at Rocketdyne. The expert system envisioned has as its primary source of rules the direct articulation of the current methods and expertise being used by NASA engineers. This articulation is being accomplished using an expert interview technique, with interviews being focused on the investigation of past SSME test data reviews. In addition, a potential second source of expertise is being explored. This expertise would be in the form of rules generated by an inductive approach based on calculations produced by a theoretical model of the SSME. Rules induced from such a model would be used to supplement the human-derived expertise component of the system.

The expert system shell package Insight2+ was selected for the initial effort in this project. It was felt that Insight2+ provided enough capability to build a usuable system. Furthermore, the simplicity and ease of use of Insight2+ has allowed an increased focus on the process of collecting and organizing the needed expertise, as compared to the use of a more complex tool, which would have required that more time be spent in the process of modelling and encoding the expertise. Evaluations will be performed later to see if there are needs which can not be satisfied by Insight2+, and if this is the case, the expert system will be translated to a more flexible and powerful expert system shell tool.

Articulating Experiential Expertise

An important consideration in attempting to capture the experiential expertise being applied to SSME data analysis is that the persons possessing the highest level of such expertise are also the persons who are busiest in current SSME data analysis. Thus methods for such expertise capture had to be designed to be efficient and streamlined, because the needed experts' time is in short supply.

The expertise collection method used in this project is a modification of the traditional expert/knowledge engineer interview, with each interview being limited in scope to discussions and explanations of some significant past SSME anomaly analysis. It was decided that these interviews would be conducted in two segments, each segment typically lasting an hour or so. This two phase interview structure has some important advantages. After the initial interview segment, the knowledge engineer can formulate some tentative rules and in this process collect further questions and inquiries which must be answered before full-fledged rule construction can be completed. During the second interview segment (separated ideally by no more than one or two days from the first segment), a more complete articulation of the expertise to be modelled can be accomplished and the communication that occurred during the first interview segment can be checked and, if necessary, corrected.

To illustrate the pace and scope of such interviews, it is instructive to consider an example from the project. An expert interview focusing on two SSME tests involving fuel leak anomalies was conducted in two segments. The fault tree derived from this interview has a maximum depth of nine, with a three level goal/subgoal structure. One of the three main branches of the fault tree is shown in Figure 1.

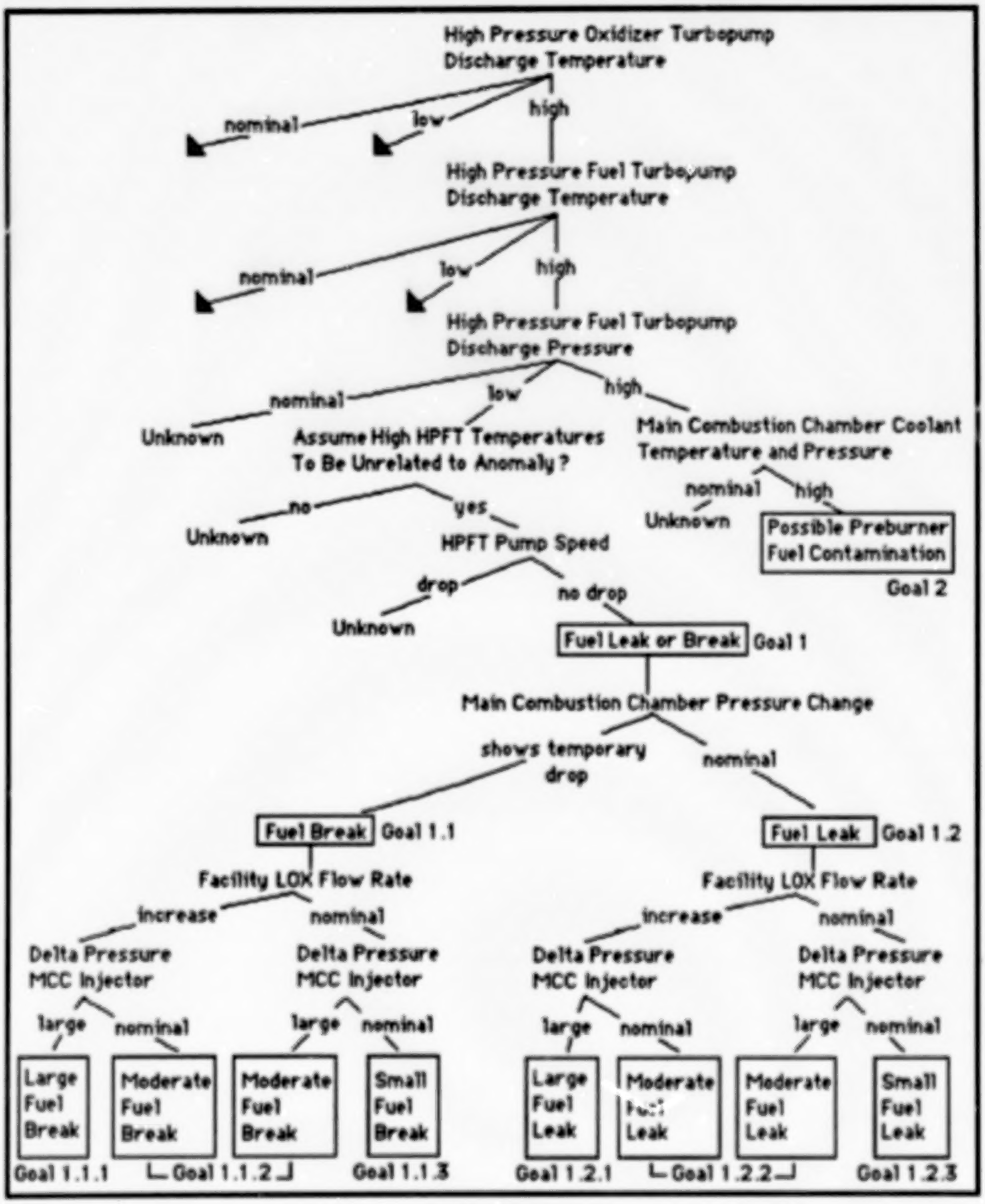


Figure 1: A Portion of the Example Fault Tree.

From the above interview, 19 rules were formulated and incorporated into the expert system rule base. Sixteen of these rules concluded some category of fuel leak, but three rules were derived for other anomalies. These other anomalies had been considered during the initial SSME data analysis for the tests under consideration and then dismissed. However, the hypothetical cases that were constructed for them before their dismissal provided the basis for the rules constructed.

In the tree in Figure 1, note the inclusion of the *Unknown* branches. These represent stubbed reasoning paths in the tree, and suggest possible directions for future expert interviews. Note also how the goal/subgoal structure is incorporated. When a goal at any level is reached, an appropriate report is given to the session user, and the user is invited to use the expert system's "explain" facility to further examine the reasoning applied to reach this goal or subgoal. This hierarchical structure has been designed explicitly to enhance the use of the resultant expert system as a training tool for novice engineers and to aid senior engineers in debugging and refining the expert system.

Expertise Inductively Generated from a Model

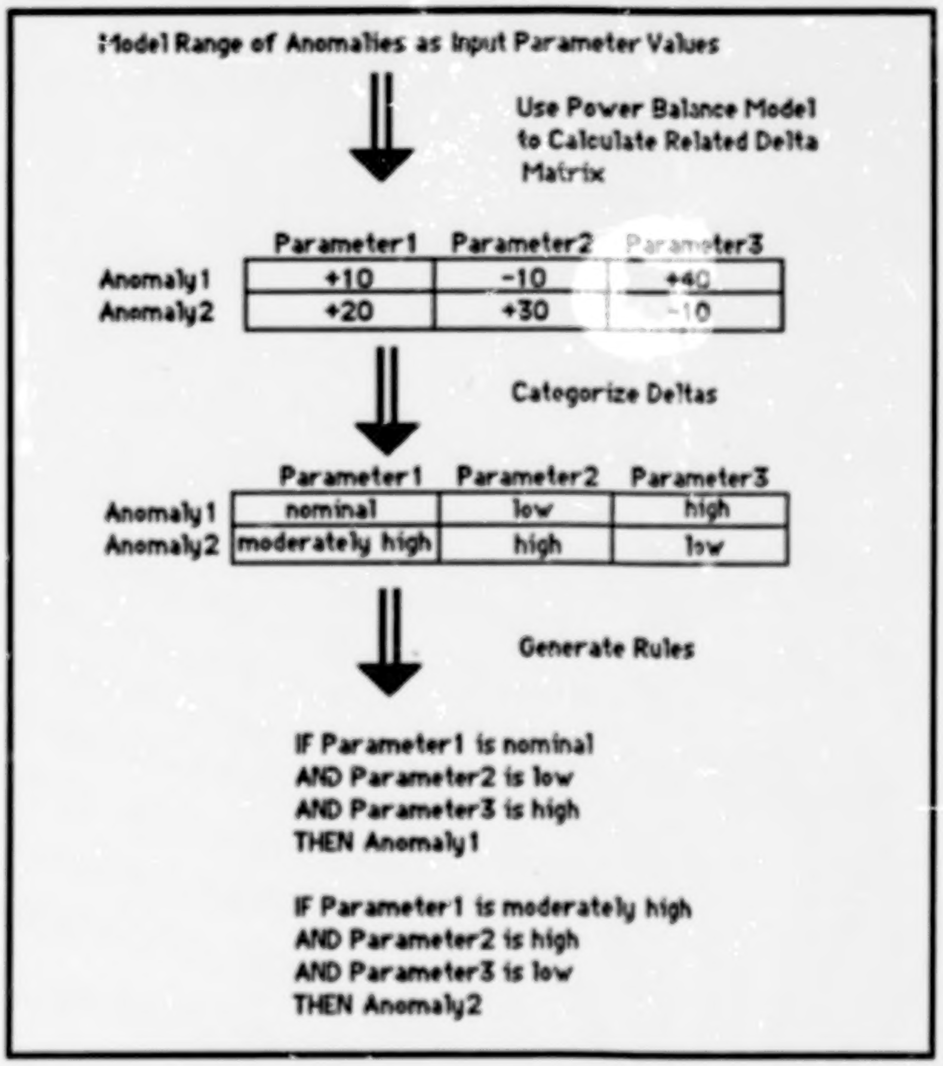
One of the limitations of the rule articulation methodology described above is that it focuses primarily on building rules to recognize anomalies similar to those that have already been observed and analyzed. Of course, it is desirable to have rules that can also recognize and categorize anomalous engine behavior of a type that is being observed for the first time. One possible way of capturing such expertise, which would complement the human-derived expertise decribed above, involves generating rules indirectly from a theoretical model of the SSME. A model called the *power balance model*, developed jointly by NASA and Rocketdyne, would appear to provide this capability. This model is already being used for direct validation of suspected anomalies. The proposed indirect use of the model for inductive expert rule generation is described below.

The power balance model takes values for some chosen parameters as input and computes changes in predicted values for other related engine performance parameters. By modelling how a hypothetical anomaly will directly affect the values of some input parameters, the power balance model can be used to calculate the predicted effect the anomaly will have on other parameters. Using the model in this way, changes (or deltas) in critical performance parameters can be calculated for a range of anomaly conditions.

For example, nozzle fuel leaks of various magnitudes could be translated into the appropriate input parameter values to produce a calculated matrix of predicted deltas for the chosen performance parameters. The calculated deltas could then be categorized, and from the matrix of categorized deltas, diagnostic rules could be formulated.

A set of rules generated using the methods described above forms a flat (or nonstructured) rule base. Existing methods and programs, [1] and [5], can then be applied to convert this rule base to structured form and to produce a goal/subgoal hierarchy. A simplified hypothetical example situation is illustrated in Figure 2. The most difficult part of this complete methodology is representing the anomalies as model input. This activity often requires some adjustments to the model itself and requires an expert.

It should be possible to write programs to automate the entire process of categorizing the deltas, generating a rule set from the categorized delta matrix, and then converting this rule set to structured form for inclusion in the expert system rule base. Once this automation is accomplished, the calculation of the delta matrices and the definitions of the various delta categories could be changed when desired and the structured rule set automatically regenerated. It is anticipated that the current expert system will be expanded using this methodology in the future.


Model Range of Anomalies as Input Parameter Values
Use Power Balance Model to Calculate Related Delta Matrix
Categorize Deltas
Generate Rules

	Parameter1	Parameter2	Parameter3
Anomaly1	+10	-10	+40
Anomaly2	+20	+30	-10

	Parameter1	Parameter2	Parameter3
Anomaly1	nominal	low	high
Anomaly2	moderately high	high	low

IF Parameter1 is nominal
AND Parameter2 is low
AND Parameter3 is high
THEN Anomaly1

IF Parameter1 is moderately high
AND Parameter2 is high
AND Parameter3 is low
THEN Anomaly2

Figure 2: Using the Power Balance Model to Generate Rules.

Conclusion

An SSME expert system data analysis manager containing directly collected human experiential expertise is being constructed. A methodology has been established for the collection of the expertise, and a significant amount of such expertise has been organized into the current version of the expert system. Particular attention is being paid to incorporating a goal/subgoal hierarchical structure within the rule base, with appropriate reports given to the system user when goals or subgoals are reached. This structure will optimize the system's usefulness for the novice engineer and allow senior engineers more easily to refine and debug the system.

Expansion of the current expert system is planned through the inclusion of rules derived inductively from calculations made using a theoretical model of the SSME's operation. The goal is to automate the generation of such an inductively derived rule set, so that the model assumptions and the definitions of the parameter categories used in the rules can be easily changed. Such rules will complement the existing expert system's human-derived rules, and the combination of rules from both these sources will provide a rule base with expanded capabilities for identifying a wide variety of engine anomalies.

Acknowledgments

This work was supported by NASA grant NAG8-646 (Marshall Space Flight Center). The author wishes to thank Chris Singer and Dave Seymour, both of the Propulsion Systems Division at Marshall Space Flight Center, for valuable consultations concerning this project.

References

1. Abernethy, K., "An Approach to Articulating Expert System Rule Bases," Proceedings of 1st International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Vol. 11, Tullahoma, TN, 1988, pp 814-822.

2. Asgari, D. and Modesitt, K., "Space Shuttle Main Engine Test Analysis: A Case Study for Inductive Knowledge Based Systems Involving Very Large Data Bases," IEEE International Conference on Computer Software and Applications (COMPSAC), 1986, pp. 66-71.

3. Daumann, A. and Modesitt, K., "Space Shuttle Main Engine Performance Analysis Using Knowledge Based Systems," ASME International Conference on Computers in Engineering, 1985, pp. 55-62.

4. Garcia, R., Jr., "An Expert System to Analyze High Frequency Dependent Data for the SSME Turbopumps," Third Conference on Artificial Intelligence for Space Applications, Huntsville, AL, 1987, pp. 213-220.

5. Kerstetter, T. and Abernethy, K., "A Program to Generate an Expert System Rule Base from a Failure Mode Model," Proceedings of 26th Annual Meeting of the Southeast Region ACM, Mobile, AL, 1988, pp 130-34.

6. Modesitt, K., "Space Shuttle Main Engine Anomaly Data and Inductive Knowledge Based Systems: Automated Corporate Expertise," Third Conference on Artificial Intelligence for Space Applications, Huntsville, AL, 1987, pp. 203-212.

N89-15607

SPACECRAFT ENVIRONMENTAL ANOMALIES EXPERT SYSTEM

H. C. Koons and D. J. Gorney

Space Sciences Laboratory
The Aerospace Corporation
Mail Station M2-260
P.O Box 92957
Los Angeles, Ca. 90009

ABSTRACT

A micro-computer-based expert system is being developed at the Aerospace Corporation Space Sciences Laboratory to assist in the diagnosis of satellite anomalies caused by the space environment. The expert system is designed to address anomalies caused by surface charging, bulk charging, single event effects and total radiation dose. These effects depend on the orbit of the satellite, the local environment (which is highly variable), the satellite exposure time and the "hardness" of the circuits and components of the satellite. The expert system is a rule-based system that uses the Texas Instruments Personal Consultant Plus expert system shell. The completed expert system knowledge base will include 150-200 rules, as well as a spacecraft attributes database, an historical spacecraft anomalies database, and a space environment database which is updated in near-real-time. Currently, the expert system is undergoing development and testing within the Aerospace Corporation Space Sciences Laboratory.

INTRODUCTION

The Aerospace Corporation Space Sciences Laboratory (SSL) is currently involved in the development and testing of a micro-computer-based expert system to aid in the objective analysis of environmentally-induced satellite anomalies. On orbit anomalies in satellite systems or subsystems occur quite often (several hundred are reported each year), and the number, frequency and severity of the anomalies are likely to grow with the inevitable increases in spacecraft complexity in the future. Spacecraft anomalies have a wide variety of causes and wide ranges of effects and severity. Design errors and inadequate quality control in parts selection and workmanship are examples of pre-flight engineering and construction errors which can lead to later anomalous behavior of components on orbit. Naturally, all mechanical and electrical components are susceptible to failure from wearout. Wearout failures and their effects can be difficult or impossible to anticipate. Many satellite anomalies result directly from improper commanding or operation (human error or ground system error). For Defense Department space systems, the possibility of hostile action must be a consideration as well. Also, recent studies have shown that adverse interactions between spacecraft components and the natural space environment can have deleterious consequences comparable in severity and frequency to those caused by any other factor. Indeed, spacecraft anomalies attributable to electrostatic discharges (just one form of environmental interaction) have been known to cause command errors, spurious signals, phantom commands, degraded sensor performance, part failure and even complete mission loss. Regardless of the severity of the anomaly, it is important to assess the cause of the problem in a timely and accurate manner so that appropriate corrective action can be taken.

Various aspects of the space environment can cause on-orbit satellite anomalies. The plasma environment (especially in geosynchronous orbit) can cause differential charging of satellite components on the surface of the vehicle. Surface charging can exceed breakdown voltages, and electrostatic discharges (ESDs) can occur with the potential to disrupt electronic components. More energetic components of the space radiation environment can penetrate and become embedded in

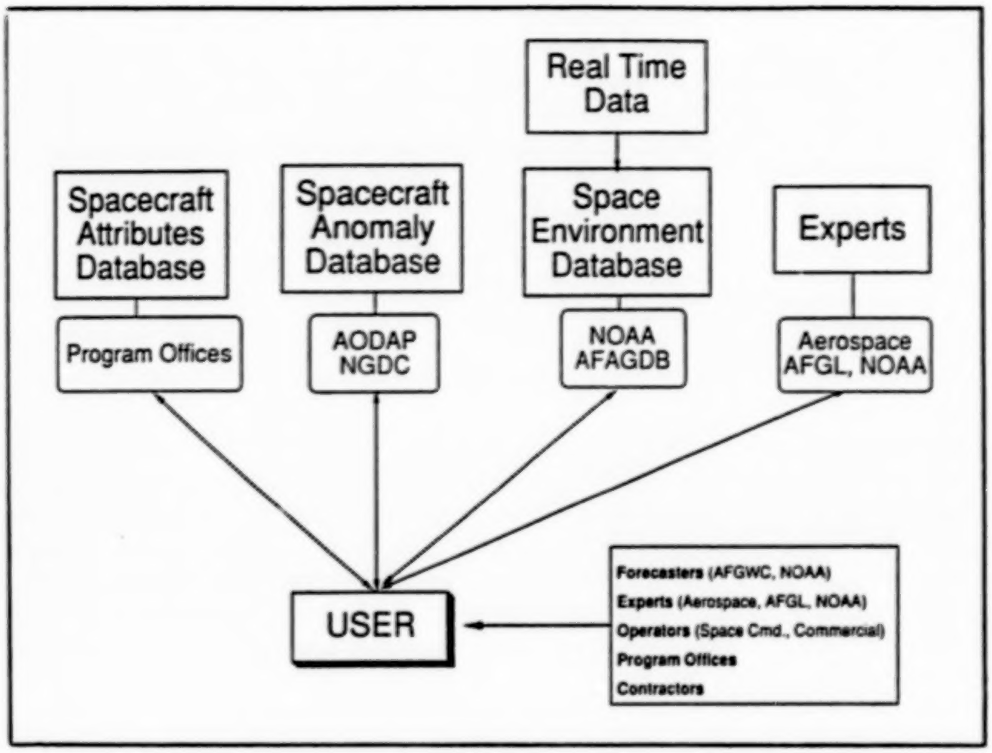

Real Time Data
Spacecraft Attributes Database
Spacecraft Anomaly Database
Space Environment Database
Experts
Program Offices
AODAP NGDC
NOAA AFAGDB
Aerospace AFGL, NOAA
USER
Forecasters (AFGWC, NOAA)
Experts (Aerospace, AFGL, NOAA)
Operators (Space Cmd., Commercial)
Program Offices
Contractors

Figure 1a

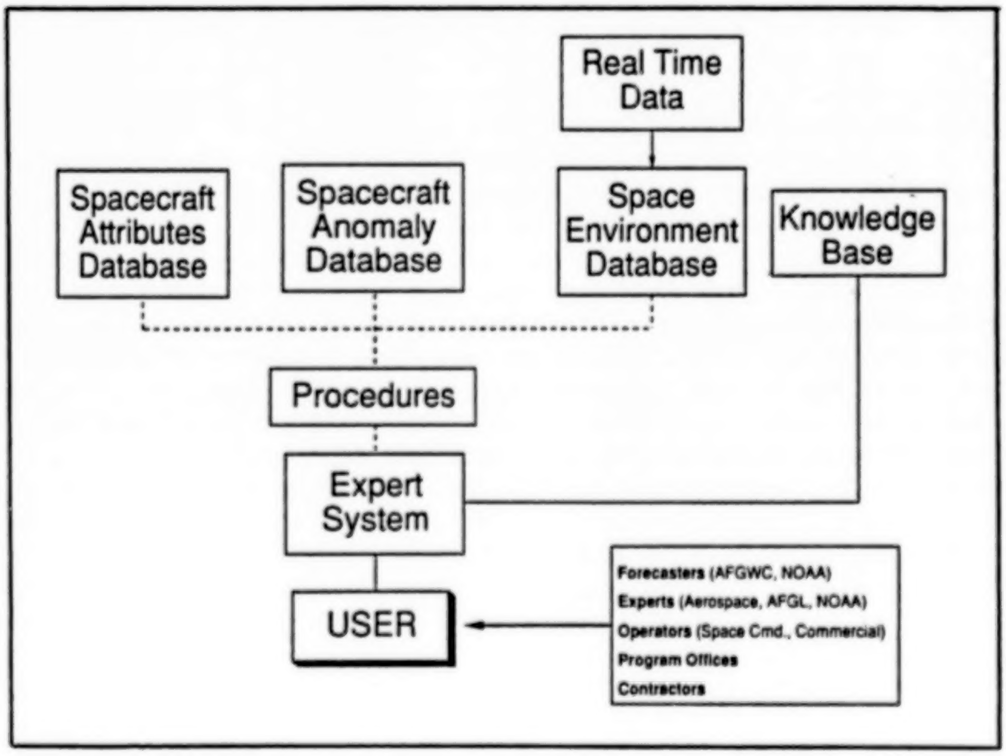

Real Time Data
Spacecraft Attributes Database
Spacecraft Anomaly Database
Space Environment Database
Knowledge Base
Procedures
Expert System
USER
Forecasters (AFGWC, NOAA)
Experts (Aerospace, AFGL, NOAA)
Operators (Space Cmd., Commercial)
Program Offices
Contractors

Figure 1b

dielectric components such as cable insulation and circuit boards. This "bulk charging" phenomenon can, under extreme circumstances, result in ESD within the dielectric components, disrupting signals or devices within the affected subsystem. Trapped radiation belt particles, solar flare protons and galactic cosmic rays can cause single event upsets (SEU) within microelectronic devices. This same high energy radiation leads to degradation of microelectronic devices and sensors through total dose effects. Other aspects of the environment, such as micrometeors and debris, can cause mechanical disruption of the vehicle. Anomalies which result from any of these environmental causes can lead to transient malfunction or even to non-recoverable loss of the component or subsystem.

Typically, various agencies and individuals become involved in the identification and resolution of spacecraft anomalies. These include (1) the commercial or military satellite operators who must evaluate the anomaly in near real time in order to take proper corrective action or to "safe" the vehicle; (2) space-environment forecasters, such as the Air Force Global Weather Central (AFGWC) or the NOAA Space Environment Services Center (SESC), who must assess the environmental situation in real time and issue warnings and alerts regarding hazardous conditions; (3) satellite contractors, who must assess the susceptibility of their vehicle and incorporate design modifications if the vehicle's on-orbit reliability proves to be inadequate; and (4) scientists and engineers who develop an understanding of the processes by which the environment interacts with the satellite, with the goal of recommending mitigating procedures for future missions. A major difficulty with the spacecraft anomaly diagnosis problem is that many of the operational groups who must quickly and accurately diagnose the anomaly do not have immediate access to the required data or to the scientific or engineering expertise required to properly assess the role of the space environment in the anomaly. Thus, real-time spacecraft anomaly diagnosis appears to be an ideal application for an expert system which can gather, format, display and utilize appropriate data consistent with logical rules based on state-of-the-art engineering and scientific expertise. The Aerospace Corporation Space Sciences Laboratory, based on its long-term involvement in space environment research and its continuous interactions with the satellite operational community, has undertaken the task of developing such an expert system.

SYSTEM DESCRIPTION

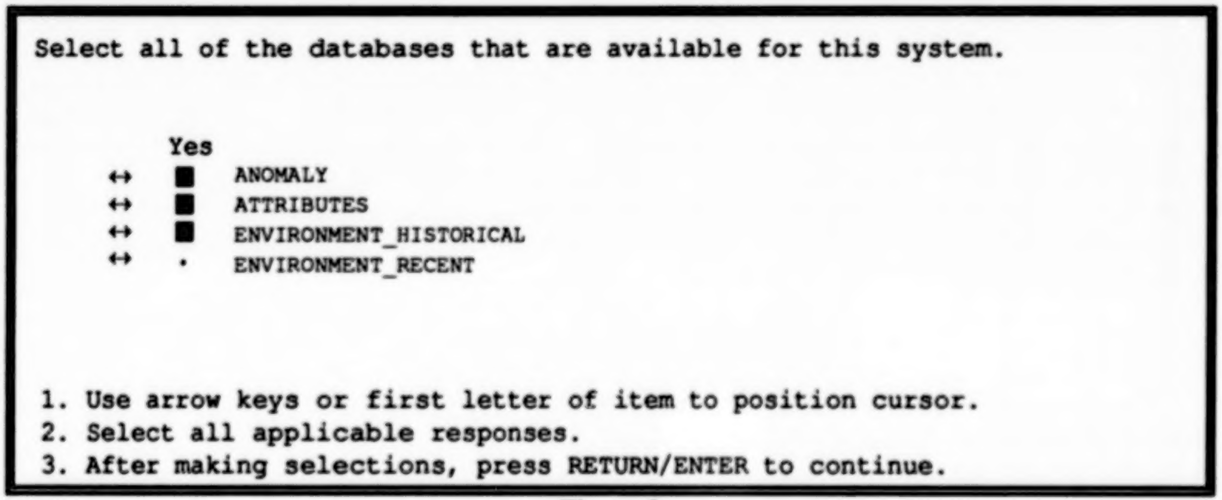
The Spacecraft Environmental Anomalies Expert System has been developed within the framework of the Texas Instrument's Personal Consultant Plus expert system shell, and has been implemented for development and test on a Compac Deskpro 386 computer. The basic architecture of the system is designed to conform as much as possible to the working environment in which the system will ultimately be implemented while taking advantage of the data-processing, data display and decision-making functions of the computer system. The design also makes use of existing data sets. Figure 1a shows the typical user environment without the benefit of an expert system. A user (anyone who is responsible for the diagnosis of a satellite anomaly) generally has access to a number of data bases and to some amount of expertise. The data bases might be computer-based or in the form of technical literature; the data bases might be on-site or at other agencies. The most pertinent data bases are the Space Environment Data Base, which contains historical and real-time data on the space environment; the Spacecraft Attributes Data Base, which contains information on the vehicles (this might include component information, ephemerides, etc.); and a Spacecraft Anomaly Data Base containing records of previous anomalies on the vehicle in question. Expert opinions are usually available over the phone or in consultations after the fact. Even under the best of circumstances the user has a formidable task to acquire and digest the information pertinent to his diagnosis. Often the user may not know what information is available or what information is pertinent.

Figure 1b shows the implementation of the Spacecraft Environmental Anomaly Expert System. The expert system not only provides access to a consolidated interactive knowledge base, but also provides procedures to access and display information from the data bases. The knowledge base, currently consisting of over 100 logical rules, has been constructed based on personal interviews with space scientists and engineers. In practice, the expert system could be operated (albeit will diminished effectiveness) even if one or more of the prescribed data bases were unavailable. At present, the expert system is configured to diagnose anomalies caused by surface charging, bulk charging, single event effects and total radiation dose. The system could be expanded to include a broader range of environmental causes of anomalies without any difficulty. The performance of the system has been tested on historical case studies, and the following section gives a typical example of a single consultation with the expert system.

SAMPLE CONSULTATION

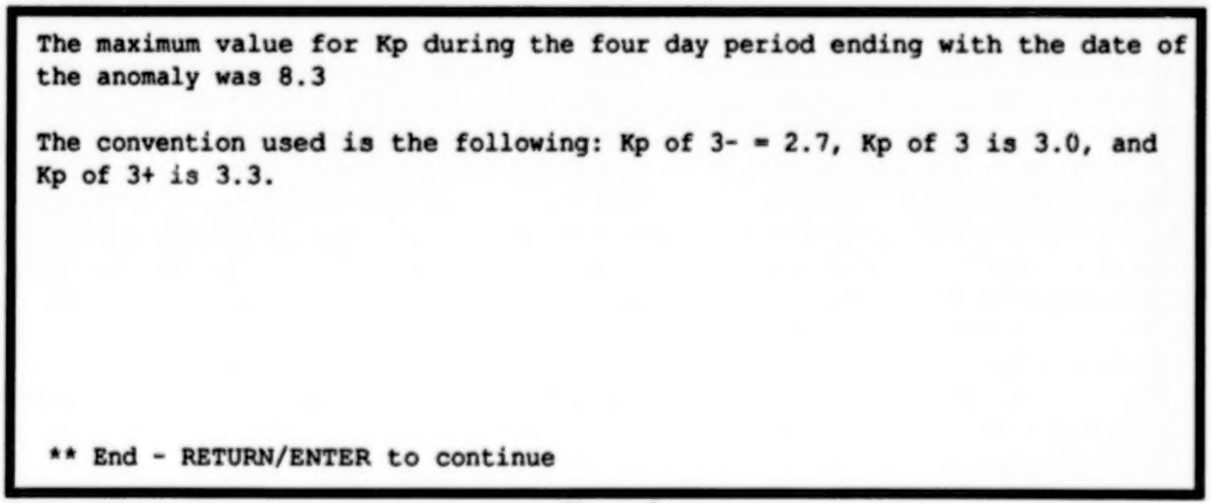
We will follow a typical consultation with the Spacecraft Environmental Anomalies Expert System by showing a selection of representative screens through which the user interacts with the system. First the user selects the Expert or Novice mode. The Novice mode contains more descriptive material and antecedent rules that explain the conclusions reached by the system. It is intended as a learning tool.

The user next selects from Fig. 2 the data bases that he wishes to use during the consultation. This also permits a flexible configuration at different sites where one or more of the databases may not be available. The four databases shown in Fig. 2 represent different techniques for storing and accessing data. The Anomaly database is a Dbase III Plus file provided by the Na-



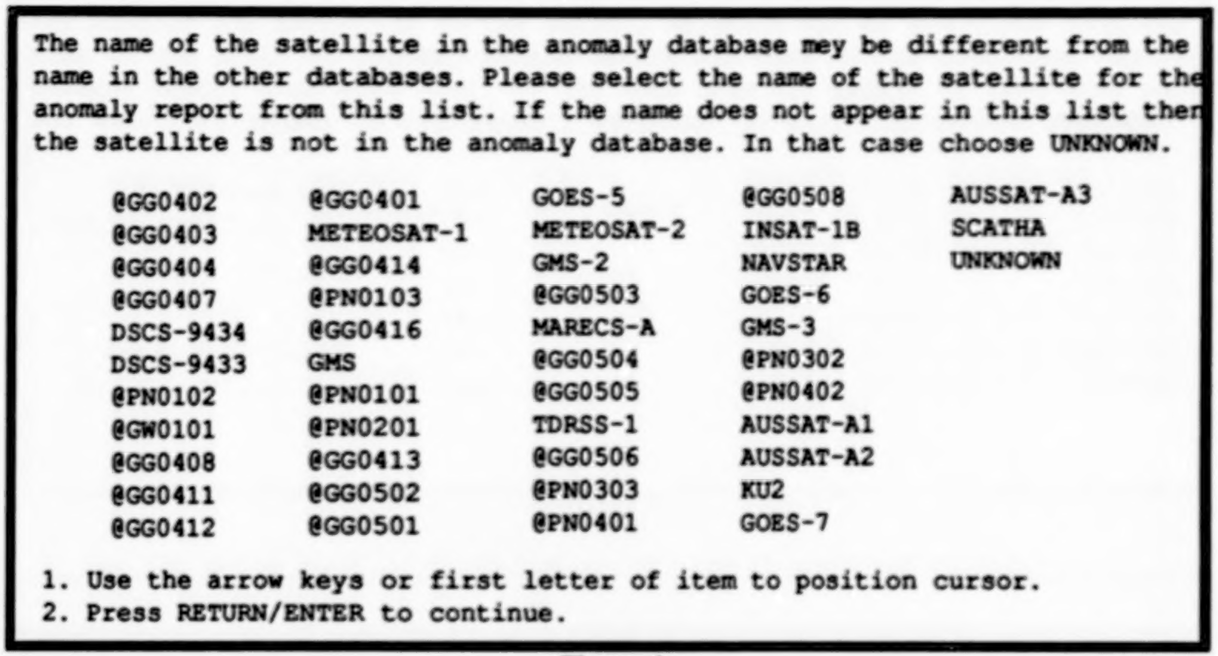
```
Select all of the databases that are available for this system.


          Yes
   ↔       ■     ANOMALY
   ↔       ■     ATTRIBUTES
   ↔       ■     ENVIRONMENT_HISTORICAL
   ↔       •     ENVIRONMENT_RECENT



1. Use arrow keys or first letter of item to position cursor.
2. Select all applicable responses.
3. After making selections, press RETURN/ENTER to continue.
```

Figure 2



```
The maximum value for Kp during the four day period ending with the date of
the anomaly was 8.3

The convention used is the following: Kp of 3- = 2.7, Kp of 3 is 3.0, and
Kp of 3+ is 3.3.








 ** End - RETURN/ENTER to continue
```

Figure 3



```
The name of the satellite in the anomaly database mey be different from the
name in the other databases. Please select the name of the satellite for the
anomaly report from this list. If the name does not appear in this list then
the satellite is not in the anomaly database. In that case choose UNKNOWN.

        @GG0402       @GG0401       GOES-5        @GG0508      AUSSAT-A3
        @GG0403       METEOSAT-1    METEOSAT-2    INSAT-1B     SCATHA
        @GG0404       @GG0414       GMS-2         NAVSTAR      UNKNOWN
        @GG0407       @PN0103       @GG0503       GOES-6
        DSCS-9434     @GG0416       MARECS-A      GMS-3
        DSCS-9433     GMS           @GG0504       @PN0302
        @PN0102       @PN0101       @GG0505       @PN0402
        @GW0101       @PN0201       TDRSS-1       AUSSAT-A1
        @GG0408       @GG0413       @GG0506       AUSSAT-A2
        @GG0411       @GG0502       @PN0303       KU2
        @GG0412       @GG0501       @PN0401       GOES-7

 1. Use the arrow keys or first letter of item to position cursor.
 2. Press RETURN/ENTER to continue.
```

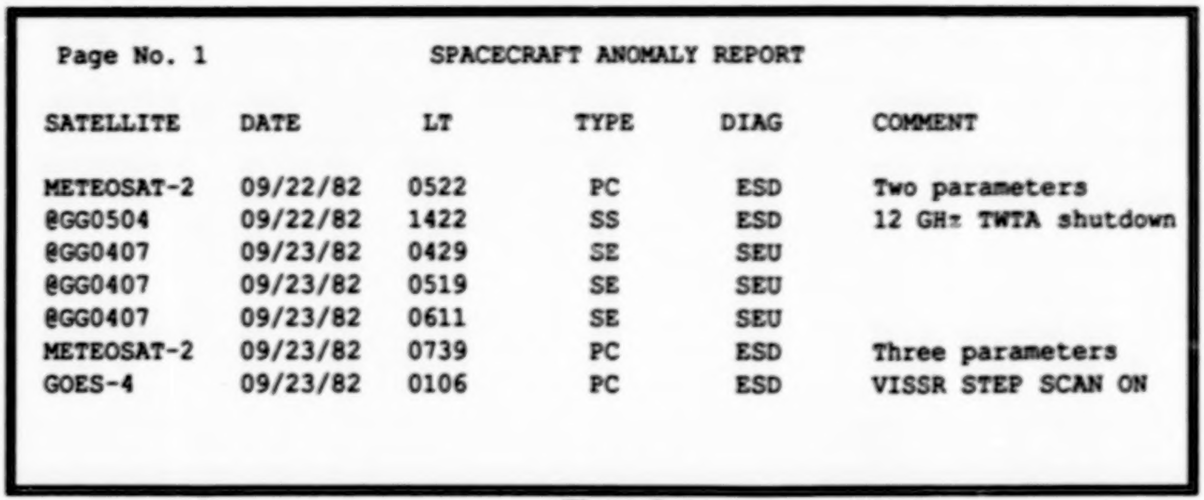
Figure 4

tional Geophysical Data Center in Boulder, CO. It presently contains a record of approximately 2000 historical anomalies. The Attributes database consists of a text (ASCII) file for each of a small selection of satellites. It contains orbital information. The Environment_Historical database is a binary file that contains an historical record of the geophysical parameter known as Kp, the planetary magnetic index. Kp is a measure of the severity of magnetic storms within the Earth's magnetosphere. This file is accessed by a C-language interface between the expert system and the binary file. An example of the output from the file is shown in Fig. 3. Kp is measured on a scale from 0 to 9. The large value, 8.3, shown in Fig. 3 means that a severe storm occurred during the time period.

The Environment_Recent database has not yet been implemented. It is planned to be a binary file similar to the Environment_Historical file. The recent data will be collected by a remote computer from the satellite broadcasts by the Space Environment Services Center, Boulder, CO. When needed the most recent data in the file will be automatically transferred via telephone modem to the consultation computer.

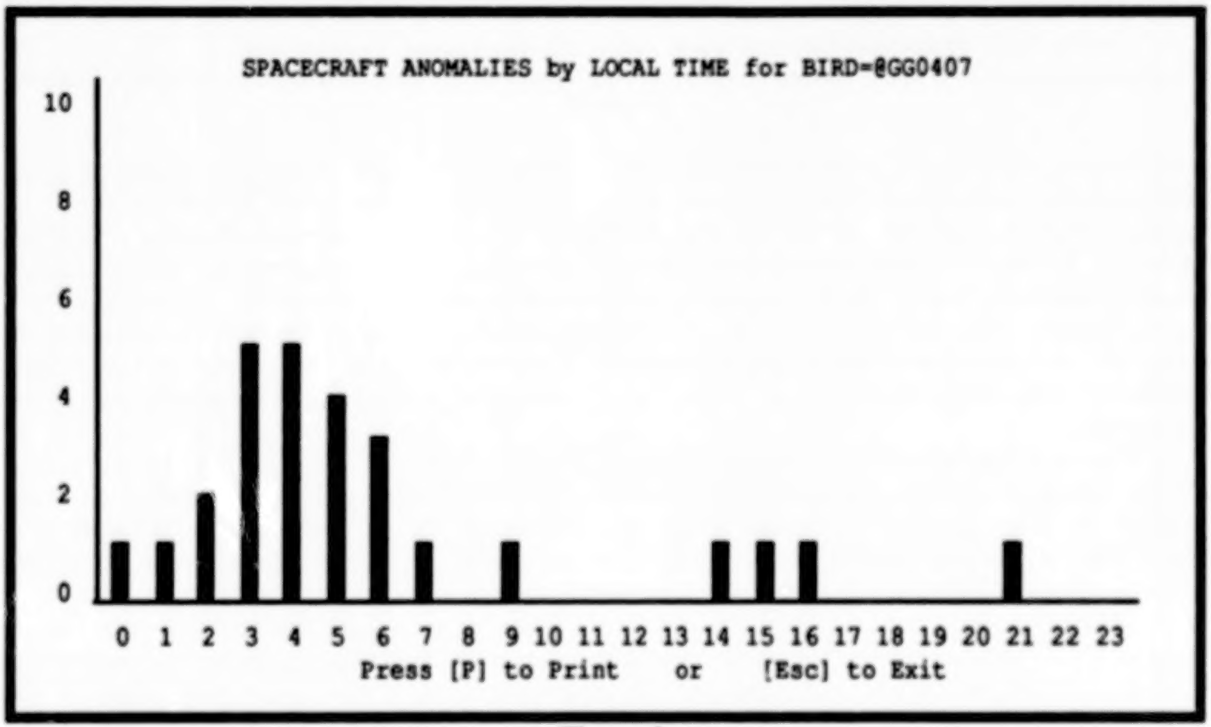
If the user chooses to use the Anomaly database the expert system automatically queries the database via a Dbase III interface and obtains a list of the satellites that are contained in the database. It then presents the satellite selection screen shown in Fig. 4 to the user. Many of the names are coded to conceal the identity of the actual vehicle. Two types of reports are generated. A satellite report lists all of the anomalies in the database for a single vehicle. This can be used to search for recurrences of similar anomalies. The recurrence in specific local time sectors or in limited regions of the orbit for example is an indication of a possible environmental cause. A data report such as the one shown in Fig. 5 lists of all of the anomalies in the database for a three day time period around the date entered by the user. Frequently more than one vehicle is affected by a severe storm. The occurrence of similar anomalies on more than one vehicle in a short time period is an indication of a possible environmental cause.

The expert system provides the user direct access to the Spacecraft Anomaly Manager software (SAM) which has been developed by the National Geophysical Data Center. The SAM utility provides a full range of functions for managing, displaying and analyzing the data, including functions to test single anomalies or sets of anomalies for environmental relationships. Histograms of local time and seasonal occurrence frequency provided by this utility can reveal distinct patterns



```
 Page No. 1                SPACECRAFT ANOMALY REPORT

SATELLITE     DATE         LT          TYPE       DIAG       COMMENT

METEOSAT-2    09/22/82    0522          PC         ESD       Two parameters
@GG0504       09/22/82    1422          SS         ESD       12 GHz TWTA shutdown
@GG0407       09/23/82    0429          SE         SEU
@GG0407       09/23/82    0519          SE         SEU
@GG0407       09/23/82    0611          SE         SEU
METEOSAT-2    09/23/82    0739          PC         ESD       Three parameters
GOES-4        09/23/82    0106          PC         ESD       VISSR STEP SCAN ON
```

Figure 5



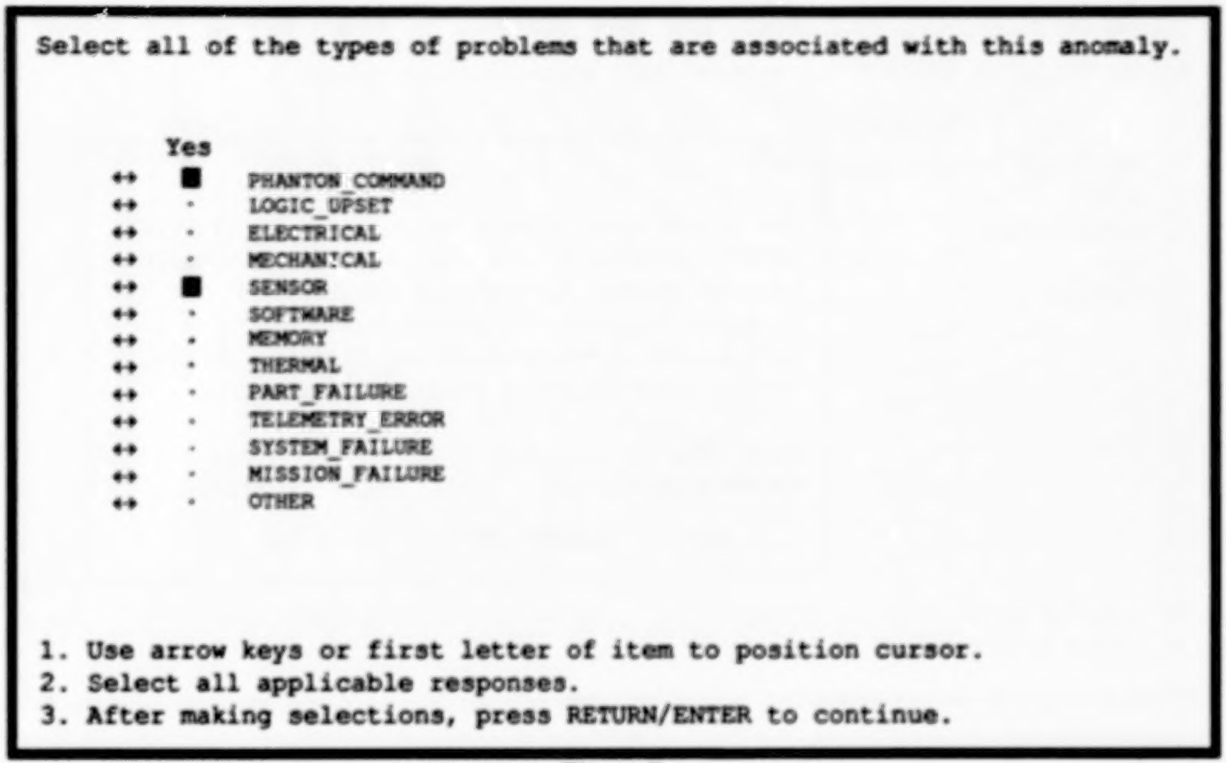
Press [P] to Print   or   [Esc] to Exit

Figure 6

for spacecraft which are susceptible to static charge buildup and ESD. Figure 6 shows one example of the graphical output available from SAM. The SAM utility also provides a means of updating the Anomaly Database after the expert system consultation.

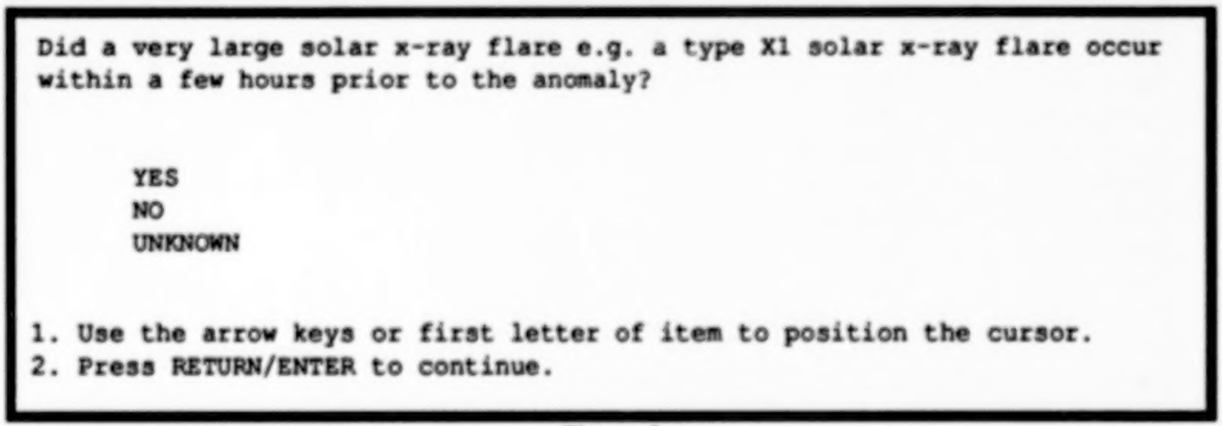
As progress is made through the consultation a number of specific questions are asked. Fig. 7 shows the screen for selecting one or more types of problems associated with the anomaly under study. The system presently has four major frames for each of the four major environmental causes of anomalies, surface charging, bulk charging, single event upset and total radiation dose. The responses to this screen are used to instantiate the frames.

Fig. 8 shows a typical question regarding the occurrence of a large solar x-ray flare. The option to answer "unknown" is always offered to the user. This may elicit another question on the same topic or cause the system to abandon that line of chaining.



```
Select all of the types of problems that are associated with this anomaly.


        Yes
    ↔   ■    PHANTON_COMMAND
    ↔   ·    LOGIC_UPSET
    ↔   ·    ELECTRICAL
    ↔   ·    MECHANICAL
    ↔   ■    SENSOR
    ↔   ·    SOFTWARE
    ↔   ·    MEMORY
    ↔   ·    THERMAL
    ↔   ·    PART_FAILURE
    ↔   ·    TELEMETRY_ERROR
    ↔   ·    SYSTEM_FAILURE
    ↔   ·    MISSION_FAILURE
    ↔   ·    OTHER


1. Use arrow keys or first letter of item to position cursor.
2. Select all applicable responses.
3. After making selections, press RETURN/ENTER to continue.
```

Figure 7

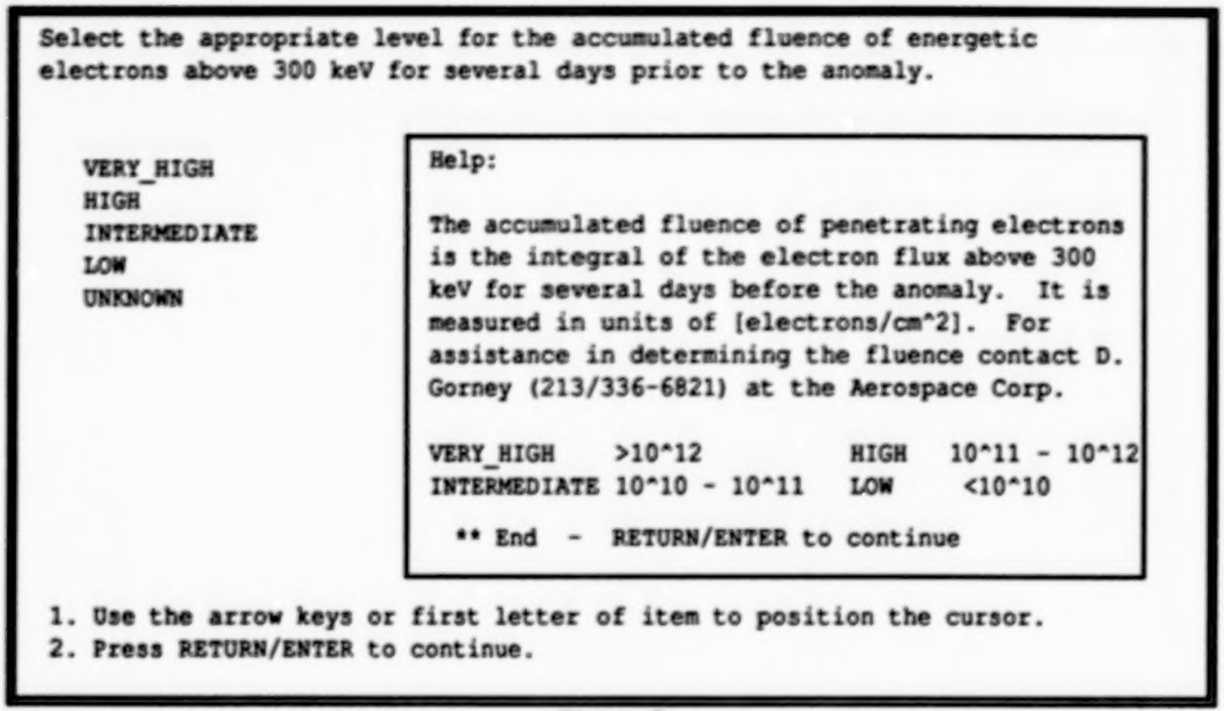


```
Did a very large solar x-ray flare e.g. a type X1 solar x-ray flare occur
within a few hours prior to the anomaly?


    YES
    NO
    UNKNOWN


1. Use the arrow keys or first letter of item to position the cursor.
2. Press RETURN/ENTER to continue.
```

Figure 8

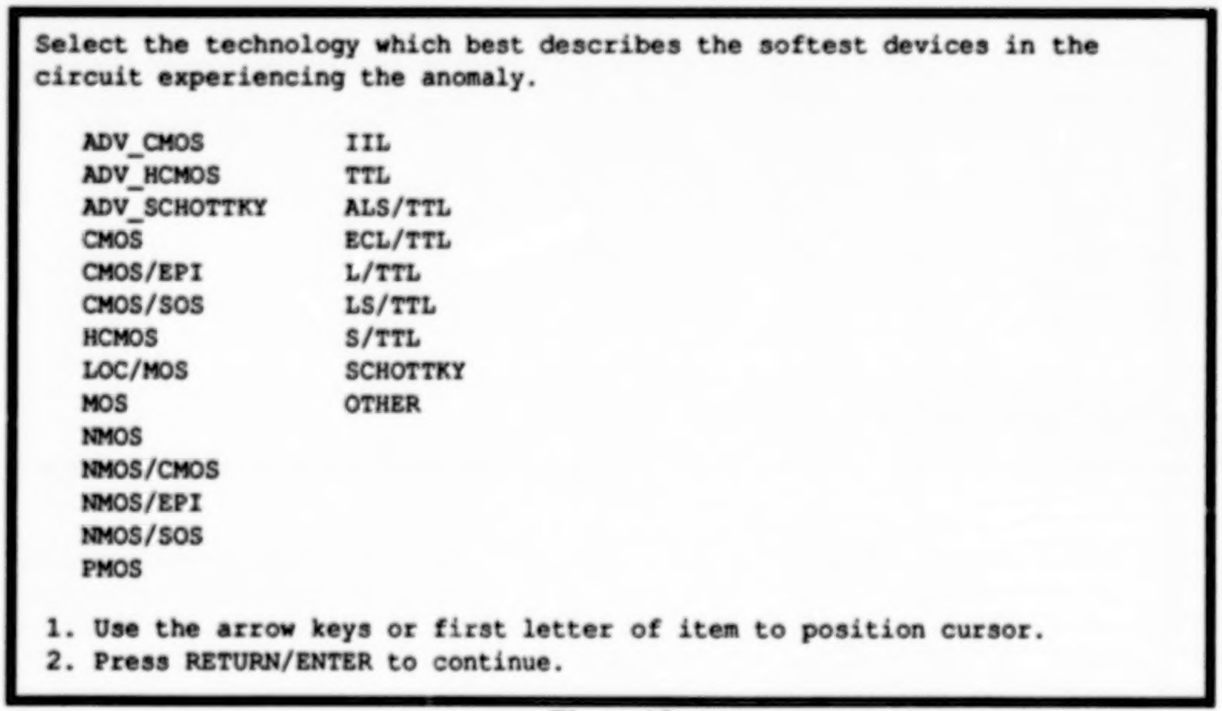
Figure 9 shows a highly technical question regarding the level for the accumulation of energetic electrons in the vehicle. This requires access to satellite environmental data plus an analysis or expert opinion to relate the measurements from the satellite making the measurements to the one experiencing the anomaly. Help is always available by pressing function key F1. The help window then appears as shown in Fig. 9. The help window contains a more detailed explanation of the question and a person or organization to contact for assistance.

Single event upsets are caused by the deposition of energy in digital devices when a very energetic particle passes through the device. The probability of an anomaly thus depends on the hardness of the device as well as on the environment. Fig. 10 shows a list of devices that is displayed for the user if the Single Event Upset frame is instantiated. A series of rules in this frame contains qualitative information on the hardness of each type of technology as determined by laboratory measurements of their upset cross sections.



```
Select the appropriate level for the accumulated fluence of energetic
electrons above 300 keV for several days prior to the anomaly.

   VERY_HIGH               Help:
   HIGH
   INTERMEDIATE            The accumulated fluence of penetrating electrons
   LOW                     is the integral of the electron flux above 300
   UNKNOWN                 keV for several days before the anomaly.  It is
                           measured in units of [electrons/cm^2].  For
                           assistance in determining the fluence contact D.
                           Gorney (213/336-6821) at the Aerospace Corp.

                           VERY_HIGH    >10^12          HIGH   10^11 - 10^12
                           INTERMEDIATE 10^10 - 10^11   LOW     <10^10

                             ** End  -  RETURN/ENTER to continue

 1. Use the arrow keys or first letter of item to position the cursor.
 2. Press RETURN/ENTER to continue.
```

Figure 9

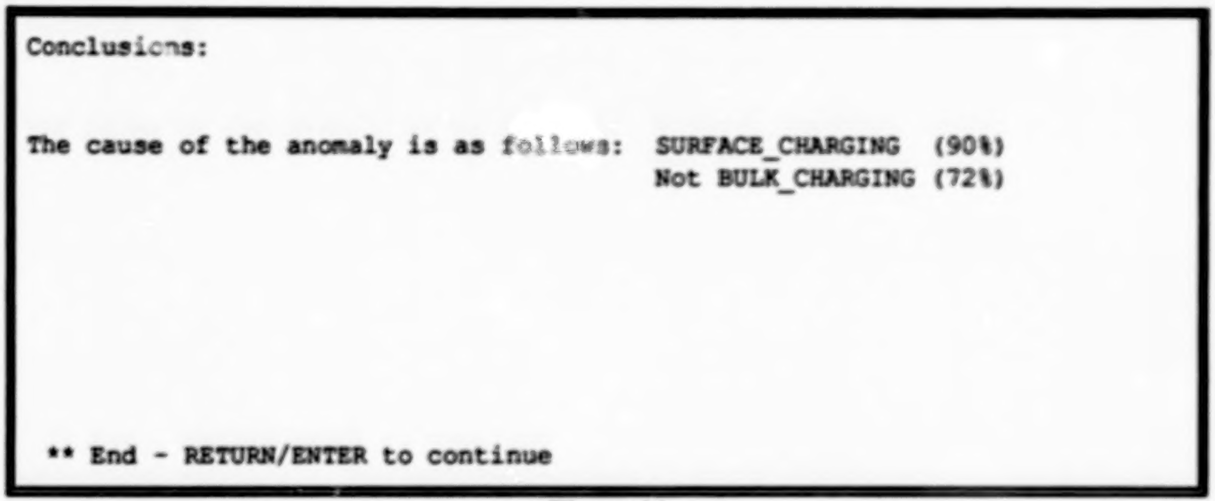


```
Select the technology which best describes the softest devices in the
circuit experiencing the anomaly.

   ADV_CMOS          IIL
   ADV_HCMOS         TTL
   ADV_SCHOTTKY      ALS/TTL
   CMOS              ECL/TTL
   CMOS/EPI          L/TTL
   CMOS/SOS          LS/TTL
   HCMOS             S/TTL
   LOC/MOS           SCHOTTKY
   MOS               OTHER
   NMOS
   NMOS/CMOS
   NMOS/EPI
   NMOS/SOS
   PMOS

 1. Use the arrow keys or first letter of item to position cursor.
 2. Press RETURN/ENTER to continue.
```

Figure 10

Finally Fig. 11 shows the conclusion screen. A confidence level is given for each possible cause. Note that a negative conclusion is listed whenever a specific cause can be ruled out.

The user may then review the consultation with the option to modify one or more of his responses to see how they affect the conclusion. Also, the u[illegible]ay choose to re-enter the Spacecraft Anomaly Manager utility to update the Anomaly Database based on the results of the consultation.



```
Conclusions:


The cause of the anomaly is as follows:  SURFACE_CHARGING  (90%)
                                         Not BULK_CHARGING (72%)







 ** End - RETURN/ENTER to continue
```

Figure 11

SUMMARY AND STATUS

The Spacecraft Environmental Anomalies Expert System is currently undergoing development and testing at the Aerospace Corporation Space Sciences Laboratory. The system aids in the analysis of satellite anomalies which may be caused by interactions with the space environment. Currently, the system deals with anomalies caused by electrostatic discharges resulting from surface or bulk dielectric charging, single event upsets, and total radiation dose. A prototype system has been developed, including three databases and a knowledge base consisting of about one hundred rules. The expert system also contains software to access, display and modify the data. The completed system will consist of approximately 150 - 200 rules. We expect that the expert system will be available for implementation at operational sites sometime during 1989. Potential users of the system include space environment forecasters at the Air Force Global Weather Central, civilian and military satellite operators, and spacecraft contractors.

ACKNOWLEDGEMENTS

We benefitted greatly from technical discussions with Drs. Joe H. Allen, Gary Heckman and Dan Wilkinson at NOAA and with Drs. J. Fennell, A. Vampola, and W. A. Kolasinski at the Aerospace Corporation. The Spacecraft Anomaly Manager software used in our expert system is provided by the NOAA National Geophysical Data Center. This project was supported by the Aerospace Sponsored Research Program and by the US Air Force System Command's Space Division under Contract No. F04701-86-C-0087 through the Satellite Control Network Program Office.

N89-15608

Using Hypermedia to Develop an Intelligent Tutorial/Diagnostic System for the Space Shuttle Main Engine Controller Lab

Daniel O'Reilly
Robert Williams
Kevin Yarbrough
Rocketdyne Division Rockwell Intl.
2227 Drake Ave. Suite 45
Huntsville Al. 35805

Abstract

This system is a tutorial/diagnostic system for training personnel in the use of the Space Shuttle Main Engine Controller (SSMEC) Simulation Lab. It also provides a diagnostic capable of isolating lab failures at least to the major lab component. The system was implemented using Hypercard, which is an implementation of hypermedia running on Apple Macintosh computers. Hypercard proved to be a viable platform for the development and use of sophisticated tutorial systems and moderately capable diagnostic systems.

This tutorial/diagnostic system uses the basic Hypercard tools to provide the tutorial. The diagnostic part of the system uses a simple interpreter written in the Hypercard language (Hypertalk) to implement the backward chaining rule based logic commonly found in diagnostic systems using Prolog.

Some of the advantages of Hypercard in developing this type of system include sophisticated graphics capablility, the ability to include digitized pictures, animation capability, sound and voice capability, and its ability as a hypermedia tool. The major disadvantage is the slow execution time for evaluation of rules (due to the interpretive processing of the language). Other disadvantages include the limitation on the size of the cards, that color is not supported, that it does not support grey scale graphics, and its lack of selectable fonts for text fields.

Introduction

The lab for which the tutorial/diagnostic was developed provides an integrated test environment for verifying the software for the Space Shuttle Main Engine Controller (SSMEC). It includes real and simulated engine hardware components. The lab software controls the hardware through several computers to allow the test engineer to force off-nominal conditions and record the reactions of the SSMEC. The central theme followed in developing the lab was that all actions and results for the tests to be conducted should be contained in a single test procedure. Additionally, the entire process should be automated to the point that the user could conduct a series of tests by entering one command to the VAX. Finally, all actions taken by the user, the lab components, and the results must be logged such that one could exactly repeat a

PRECEDING PAGE BLANK NOT FILMED

test at a later date. Under these provisions many of the machinations of the lab remain invisible to the user. This tutorial and diagnostic was designed to help lab users to understand the lab and isolate problems in the lab.

The Tutorial/Diagnostic

The prototype tutorial/diagnostic system was initially implemented using Turbo Prolog on an IBM PC and later implemented using Hypercard on a Macintosh (Hypercard is Apple's implementation of hypermedia). In ease of development, particularly in the tutorial portion, Hypercard proved to be easier and faster to use than Turbo Prolog. The Hypercard version includes extensive graphics, some animation, and some sound.

The system addresses four major areas; the use of the tutorial/diagnostic, conducting tests in the lab, the hardware operation of the lab, and the diagnostic. The part illustrating lab operations has not yet been completed. Four buttons on the top card of the stack control entry into each of these areas. When the user selects one of the buttons, this top card is "pushed" so that is may later be "popped" in response to clicking a "return" button. Figure 1 illustrates the top level layout.

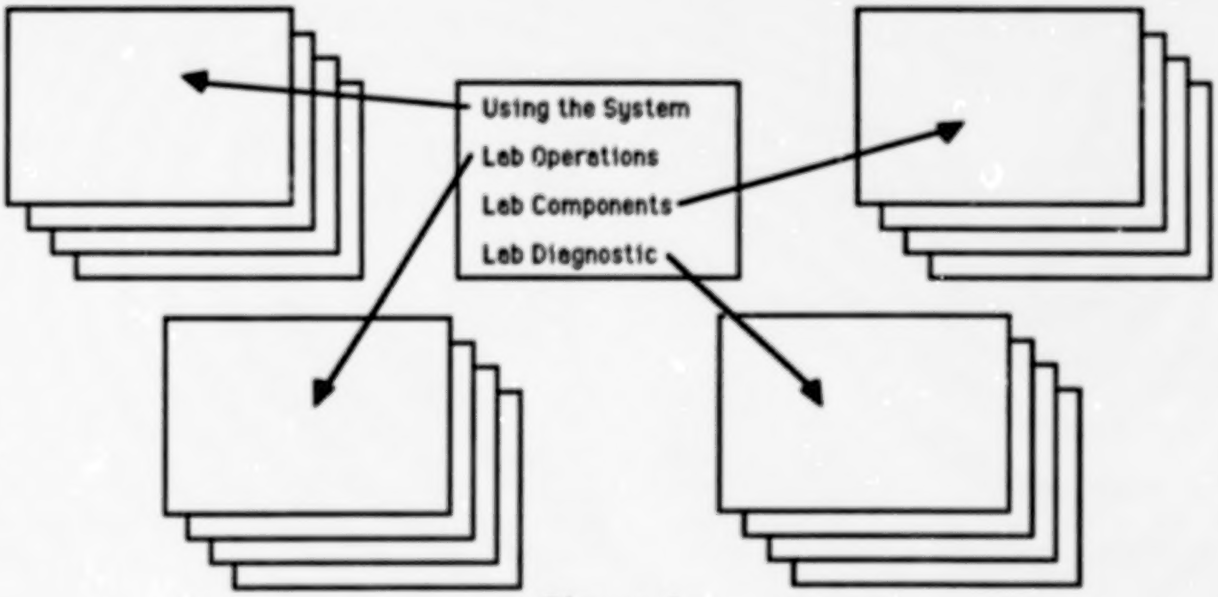


Figure 1
Top Level Layout

The first area instructs the user in using the tutorial/diagnostic. It attempts to illustrate rules the user can use to recognize buttons, navigate through the system, and get more information on a subject. To implement this area the developers used only the most basic capabilities of Hypercard, namely, graphics, text, buttons, push and pop, and linking cards. This area has no links to the other three areas to prevent the first time user from getting "lost" in the stack. Instead, copies of cards from the other areas are used to illustrate the system.

The area providing the tutorial on lab components consists of a main path which includes a brief description of each of the major components. Each of the descriptions of major components provide two side paths; one to a more detailed operational description of the component, and the other to a detailed hardware description of the component. As in the first area, only the most basic Hypercard capabilities were used. Figure 2 illustrates the layout of this area.

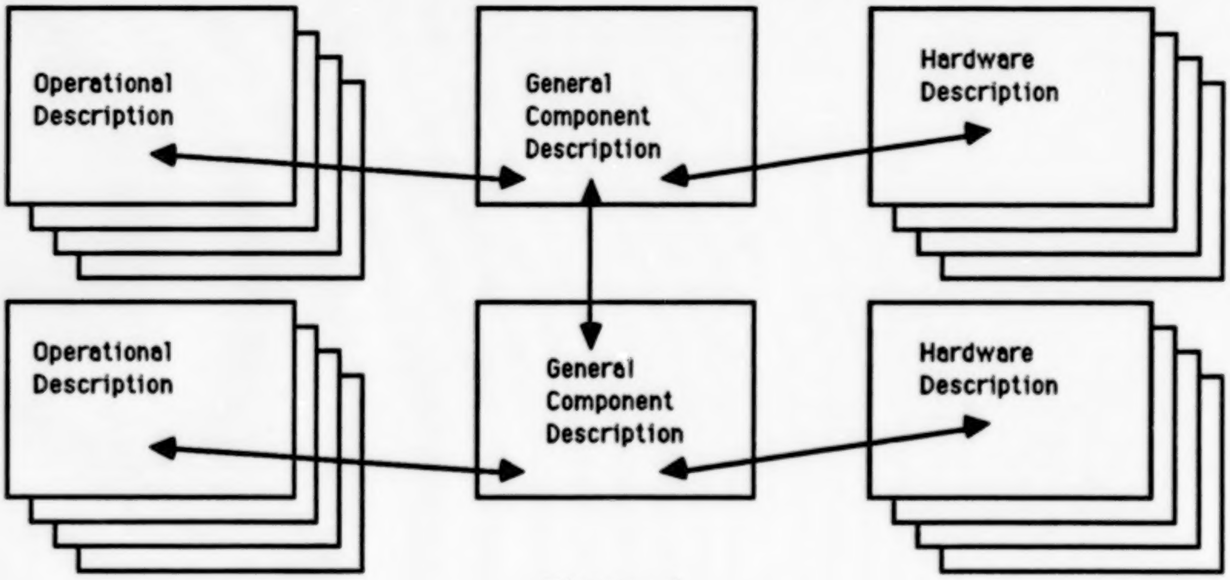


Figure 2
Lab Component Description Layout

Three basic Hypercard linking mechanisms were used to control navigation through the cards for these first three areas; direct linking of cards, go to next or previous card, and push and pop.

Direct linking of cards displays another card in response to a user action such as clicking on a button. The developers used this mechanism to go to subsets of cards from a card with multiple options. For instance, the top card of the stack gives the user four options via buttons. Clicking one of these buttons causes Hypercard to go to the first card of the subset for that option. This mechanism was also used for "help" functions where the user clicks on a button to acquire more information about a component or a test.

Go to next or previous card provides a mechanism for the user to move backward or forward to adjacent cards. This mechanism was used to allow the user to move freely among cards of a subset.

Push and pop pushes a card to the stack or pops a card from the stack. The developers used this mechanism to allow the user to return from a subset to the card at which he chose the subset. For example, when the user clicks on the button to choose a subset, the current card is pushed. The cards of the subset each have a button for "return". When this button is clicked, the card on top of the stack (in this case, the card from which the selection was made) is popped. Since push and pop are an implementation of the familiar stack operators, this mechanism may be used to "nest" this return capability.

Through the use of these basic capabilities the developers built in an orderly navigation scheme through the cards for the tutorial parts of the system. One should note that these capabilities may also be used to add implicit logic to the system in that they can be used to implement trees. Figure 3 illustrate the basic linking methods.

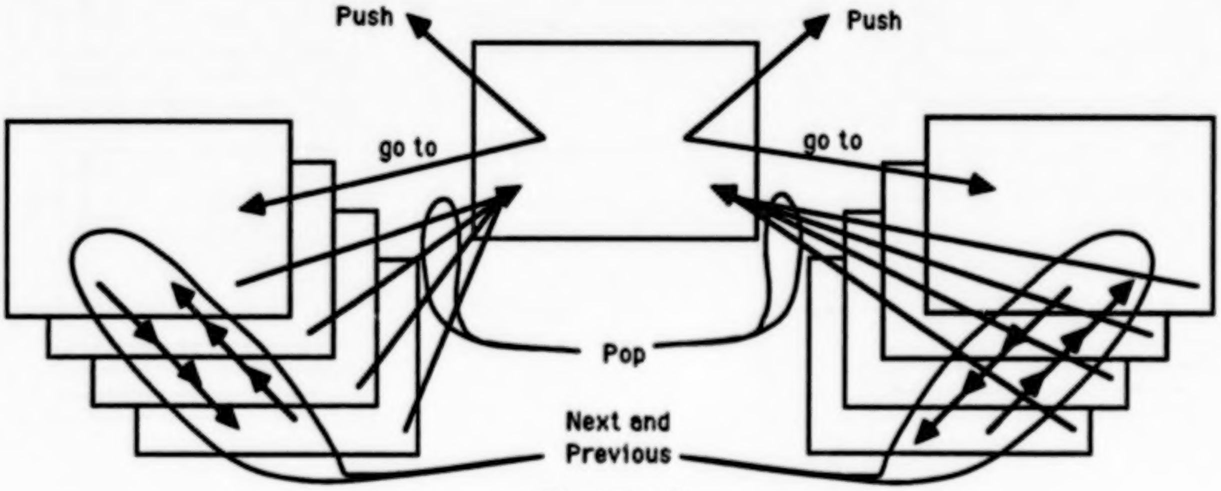


Figure 3
Basic Linking

The diagnostic comprises the fourth part of the system. This part allows the user to isolate bad components in the lab by answering a series of questions as to the symptoms exhibited at various points in the lab. Initial attempts at implementing this part included using a straight tree structure and then a modified decision tree. Ultimately, the Hypertalk capability of Hypercard was used to implement a Prolog-like rule based logic typical of many diagnostic systems.

In the first attempt, the straight tree structure quickly became too large to manage due to combinatorial expansion. In addition to becoming difficult to manage, it required the duplication of many of the symptom cards. Thus, for a system that involves any complex interrelationships or that is of any size, this method proves too cumbersome.

The next iteration attempted to simplify the tree by "modularizing" some intermediate tests and calling them from nodes in the tree. To implement this scheme the push and pop operators provided a mechanism to return to the node in the tree which called the intermediate test in order to continue the diagnostic. However, this scheme proved lacking in that useful intermediate tests were difficult to define due to the interrelationships of the lab components.

The solution involved writing a script using Hypertalk to emulate the backward chaining rule based logic typically used in developing diagnostic systems with Prolog. This scheme proved to be relatively simple to design and use and has the added advantage of providing a shell that could be used for any diagnostic system.

The diagnostic uses four basic types of cards. These include the beginning card, test cards, conclusion cards, and symptom cards. Scripts at the stack level record the results of symptoms and tests, evaluate tests, and provide the navigation among the cards.

The beginning card serves as an introduction to the diagnostic. When the user selects "continue", the script for the "continue" button sends a message to the handler that initializes variables for the diagnostic session and pushes the first test onto the test stack. Figure 4 illustrates the layout of the beginning card.

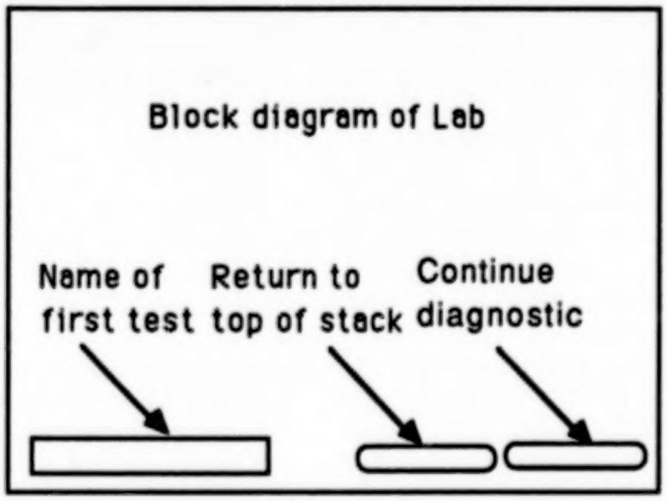


Figure 4
Layout of Beginning Card

Test cards actually define tests and may themselves be symptoms for other tests. These cards contain a description of the test in the language processed by the evaluation script. This language basically allows the knowledge engineer to describe a test in terms of boolean functions. The knowledge engineer enters this description into background field 1 which covers the upper 2/3 of the card. The outline of the test is basically an "if-then-else" statement where the conditions to be evaluated lie between the if and the then keywords. Parentheses may be used to control evaluation. Since tests themselves evaluate to true or false, they may be used in the description of other tests. However, all tests must eventually reduce to a set of symptoms. If the user inadvertently generates an endless loop (for example: making test A dependent on test B which is dependent on test C which is dependent on test A) the diagnostic, at run time, will post an error in the message box. Test cards allow the user the options to continue the diagnostic or to return to the beginning card. If the user elects to continue the diagnostic, the script for that button sends a message to push the name of the test card on the test stack and then sends a message to evaluate the test at the top of the test stack. Figure 5 illustrates the layout of a test card.

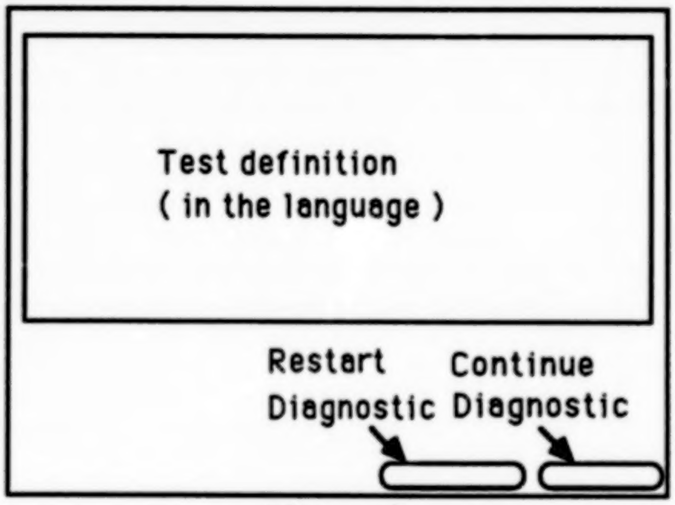


Figure 5
Layout of a Test Card

Symptom cards ask the user to enter the status of some test point in the lab. The user selects a button labelled "good" or a button labelled "bad" to indicate status at that point. Clicking the button sends a message to a handler at the stack level which records the symptom and its status. It then requests the evaluation script to evaluate the test. In addition to acquiring results, the symptom cards also allow the user to select, via buttons, more information on the lab components and the test being performed. Figure 6 illustrates the layout of a symptom card.

Conclusion cards provide the message identifying the bad component if one is found. Note that a card stating that no bad component could be identified is also a conclusion card. The knowledge engineer enters these card names in the "true" path of the test description on a test card. When the test proves true, this card is displayed. The conclusion card allows a user the choice of continuing the search for bad components or returning to the beginning card to begin a new session. Figure 7 illustrates the layout of a conclusion card.

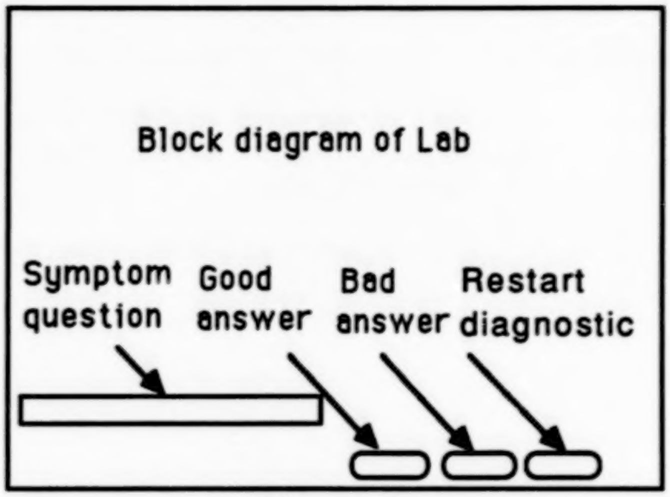


Figure 6
Layout of a Symptom Card

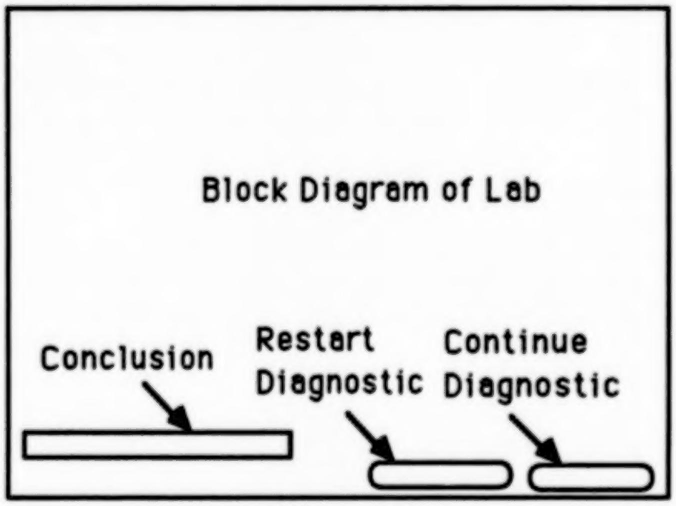


Figure 7
Layout of a Conclusion Card

The evaluation script evaluates the language on the test cards to determine the result and the action to take. This script consists of a simple parser to evaluate an "if-then-else" statement and provide branching to other cards as necessary or as dictated by the results of the "if". The following is the BNF for the test language.

```
<test_card_statement>     ::= 'if' <conditional> | 'ifopt'
                          <conditional>

<conditional>             ::= <expression> 'then' <action> 'else'
                          <action>

<expression>              ::= <simple_expression> |
                          <parenthetical_expression>
```

```
<parenthetical_expression>   ::= '(' <expression> ')'

<simple_expression>          ::= <status_card_id> <status> |
                                 <status_card_id> <status> <operator>
                                 <expression>

<status_card_id>             ::= <test_card_id> | <symptom_card_id>

<status>                     ::= 'good' | 'bad'

<operator>                   ::= 'or' | 'and'

<action>                     ::= <test_card_id> |
                                 <conclusion_card_id> | 'return'

<test_card_id>               ::= the name of a test card (there must
                                 be no intervening blanks)

<symptom_card_id>            ::=  the name of a symptom card (there
                                 must be no intervening blanks)

<conclusion_card_id>         ::= the name of a conclusion card
                                 (there must be no intervening blanks)

NOTE: white space must separate all tokens
```

The language offers two versions of the if statement: the 'if' and the 'ifopt'. The parser evaluates the 'if' version completely, regardless of whether the outcome can be true or false. The 'ifopt' version ceases evaluation as soon as the outcome can be determined. For example, in the statement 'if A and B and C', if A is false the entire statement evaluates false. Therefore, under the 'ifopt' version, the evaluation would cease and the 'else' path would be chosen.

The names of the cards must be used in the text describing the conditional part of the test. When the evaluation script encounters an identifier (card ID), it examines the status list to find out if the status of the test or symptom has already been determined. If so, it continues the evaluation. If not, it pushes the name of the card containing the current test and performs a "go to" the card name for which the status is unknown. If that happens to be a test card then that card is displayed and when the user chooses "continue" that card name is pushed on the test stack and a message is sent to evaluate the test on top of the test stack. If the card with the unknown status is a symptom card, that card is displayed and the user chooses "good" or "bad" for that symptom. The script for the "good" or "bad" buttons sends a message to the status message handler to record status and then sends a message to evaluate the test on top of the test stack. The card names direct navigation through the diagnostic and the evaluation script really makes no distinction between test cards and symptom cards.

Likewise, card names must be used for the "action" part of the test description. If the conditional part of a test description on a test card evaluates true the evaluation script performs a "go to" the name of the card following the "then" keyword. If the evaluation proves false the evaluation script performs a "go to" the card name following the "else" keyword. These cards may be test cards or conclusion cards. The "return" keyword presents an exception in that when it is encountered, the evaluation script sends a message to perform an evaluation of the test on top of the stack.

In the following example, all names in the conditional part of the if statement are test card names and both names in the action part of the test point to another test card. In this example the evaluation would continue at the test called "NextTest" regardless of the outcome of the tests named in the conditional part. The user could prevent this if any of the tests in the conditional part found a bad component by electing to restart the diagnostic.

```
if SSMECScaling good or
DPM good or
ADC good or
Hardware good or
GainDAC good or
OffsetDAC good or
HardwareSIASwitch then
NextTest else
NextTest
```

In the following example, the conditional part of the test description names only symptoms. In this example, the evaluation would branch to the conclusion card "SSMECScalingBad" if the results of the test were true and would return to evaluate the previous test if the result was false. In this example all of the conditional part of the test consists of symptoms. Also, notice that since the "ifopt" keyword was used, not all symptoms would necessarily be evaluated. For example, if the symptom "OffsetVDT" proved good, the "else" path would be taken since the entire conditional must be false. However, since the status of "GainVDT" had already been evaluated, its status would be recorded and that symptom card would never show up again during this session. Note also that if the evaluation reached "GainDPM" and "GainDPM" was good, the evaluation would terminate and take the "then" path since that was all that was required at that point to make the entire statement true.

```
ifopt GainVDT bad and
OffsetVDT bad and
LocalPotVDT bad and
( GainDPM good or
OffsetDPM good or
LocalPotDPM good ) then
```

```
SSMECScalingBad else
return
```

The fact that card names are used to direct the evaluation allows the system to function as the basis for any diagnostic type application. The evaluation script really knows nothing of the target system, but merely evaluates the boolean expression on the test cards and uses the card names to direct its action. Thus, this system could provide a shell for the development of prototypes for other diagnostic systems.

Conclusions

Excluding the two false starts, the diagnostic part of this system required about 50% of the effort that was spent on the Turbo Prolog version. The majority of this gain was due the the ease of implementing the tutorial and graphics parts. In these areas, the development gains afforded by Hypercard could range from 50-90% over a similar system developed using Turbo Prolog, depending on the amount of graphics used. Also, Hypercard's ability to include digitized photographs on the cards represents a significant advantage in developing tutorial systems. Another significant advantage of using Hypercard, at least for developing tutorial systems, is that the developer need not be a programmer to develop a successful system.

There are, however, some shortcomings in Hypercard. The execution time for the evaluation of the rules in the Hypercard system is much slower than that in Turbo Prolog. This fact is masked somewhat by the difference in the speed of the graphics of the two systems. Other shortcomings include the limitation on the size of the cards, that color is not supported, that it does not support grey scale graphics, and its lack of selectable fonts for the text in fields.

Overall, Hypercard provides a useful tool for developing sophisticated tutorial systems and moderately sophisticated diagnostic systems. Its ability to easily combine graphics, text, digitized photographs, animation, and sound, as well as its ability to function as a hypermedia tool makes it very powerful for developing tutorial systems. These capabilities also offset some of its limitations when developing diagnostic systems where these functions would prove useful.

N89 - 15609

OBJECT-ORIENTED FAULT TREE EVALUATION PROGRAM FOR QUANTITATIVE ANALYSES

F. A. Patterson-Hine
NASA Ames Research Center
MS 244-4
Moffett Field, CA 94035

and

B. V. Koen
The University of Texas at Austin
Dept. of Mechanical Engineering
ETC 5.134
Austin, TX 78712

Abstract

Object-oriented programming can be combined with fault tree techniques to give a significantly improved environment for evaluating the safety and reliability of large complex systems for space missions. Deep knowledge about system components and interactions, available from reliability studies and other sources, can be described using objects that make up a knowledge base. This knowledge base can be interrogated throughout the design process, during system testing, and during operation, and can be easily modified to reflect design changes in order to maintain a consistent information source.

An object-oriented environment for reliability assessment has been developed on a Texas Instruments (TI) Explorer LISP workstation. The program, which directly evaluates system fault trees, utilizes the object-oriented extension to LISP called Flavors that is available on the Explorer. The object representation of a fault tree facilitates the storage and retrieval of information associated with each event in the tree, including tree structural information and intermediate results obtained during the tree reduction process. Reliability data associated with each basic event are stored in the fault tree objects. The object-oriented environment on the Explorer also includes a graphical tree editor which was modified to display and edit the fault trees.

The evaluation of the fault tree is performed using a combination of standard fault tree reduction procedures. A bottom-up procedure is used for subtrees that do not contain repeated events, and a top-down, recursive procedure is used to evaluate subtrees that do contain repeated events. The tree is dynamically modularized according to the event which is being evaluated at the time. The locations of repeated events are propagated up the tree and stored in each event object. This information is used to determine which evaluation procedure is required for each event, and intermediate results are stored as they are calculated. Unlike most conventional fault tree evaluation codes which calculate the probability of occurrence of the top event only, this program produces results for every event in the fault tree. The object-oriented approach to fault tree reduction greatly increases the efficiency of the evaluation algorithms.

Introduction

Fault trees are a widely accepted method for modeling complex systems in reliability analyses. A fault tree is a graphical representation of the logical interrelationships among the components of a system [1]. The functional relationships are established in a top-down manner and are represented by logic gates such as AND- and OR-gates. A probability of occurrence is assigned to each base event denoting the failure probability of the components in the system. A fault tree can be evaluated quantitatively to determine the probability of occurrence of the top event in the tree.

The most popular method for quantifying fault trees uses the minimal cut sets of the tree and the component failure information. Minimal cut sets are sets of basic events in which each component in the set must fail for the top event to occur [2]. Algorithms which find the minimal cut sets of a fault tree are better suited to conventional programming languages than are algorithms which directly evaluate the fault tree quantitatively. Quantification of minimal cut sets produces an approximate result in most practical applications, however, since larger trees require the use of truncation techniques to restrict the number of cut sets which are generated.

Object-oriented programming languages enable algorithms which directly evaluate fault trees to be implemented easily and efficiently. These languages provide powerful features which are not available in conventional programming languages. The basic entity in object-oriented programming is the *object*, which is an abstract data type [3]. Objects are defined in terms of *classes* that combine the behavior and state of the objects. Classes describe one or more similar objects, and a particular object is called an *instance* of the class. All data and actions associated with an object are contained in the object's *instance variables* and *methods*. The instance variables represent a private memory for the object, and the procedures, or methods, associated with the object are the only legitimate operators for the data stored in the instance variables. Methods are invoked by sending a *message* to an object, which replaces conventional function calls. A unique feature called *inheritance* allows pieces of code to be shared by several objects. This eliminates unnecessary redundancy in programming and data storage. Instance variables can be accessed via the inheritance hierarchy, so changes that affect the state of all objects of a particular class can be made by updating the class definitions, thus changing at once the information inherited by multiple objects. Inheritance is not provided by any conventional language, and is quite useful in the fault tree application.

The performance of existing codes that evaluate fault trees are limited by the data structures that are available in conventional programming languages. Object-oriented programming provides the flexibility that is needed in defining structures to represent logic gates and basic event data, and the complex interrelationships which exist among them. One of the most serious problems with previous direct evaluation codes is the loss of information about the system, both pre-defined structural relationships and intermediate results during tree simplification. The application of object-oriented concepts to fault tree analysis results in a clear and concise representation of the tree and provides a powerful mechanism for the storage, retrieval, and evaluation of system information.

Fault Tree Object Definitions

This application has been developed on a Texas Instruments Explorer LISP workstation using the object-oriented extension to LISP called Flavors [4]. Logic gates and basic events share many characteristics which can be described by a general flavor, called TREE. Instance variables that contain information common to both basic events, called nodes, and logic gates are defined in this flavor. All gates and basic events have both a name and at least one parent, so two instance variables, :name and :parent, hold this information. Since the top event of the fault tree does not have a parent, the value of its :parent variable is nil. A variable called :unavailability stores probability data for basic events. Results from the simplification of logic gates is stored in :unavailability for gates. Another variable, :dependent, indicates whether repeated events are located under a particular gate. Basic events are terminal leaves, making the value of the :dependent variable nil for all basic events. The TREE flavor is specialized into two other flavors, GATE and NODE, which contain information particular to logic gates and basic events, repectively.

The GATE flavor includes instance variables that describe more specifically the state of logic gates. All gates are non-terminal leaves, and the names of their children are stored in a variable called :children. A second variable, :dependent-eval, indicates whether or not the gate must be evaluated using the top-down algorithm capable of handling repeated events. The NODE flavor inherits the :unavailability variable from the TREE flavor, but may change the value of :unavailability for all nodes by using a default specification. The :unavailability of each node can also, of course, be changed individually. Specific logic gate types, such as AND- and OR-gates, are described by flavors that are specializations of the GATE flavor. These flavors are necessary for defining methods that apply the specific reduction equations required by each type of gate.

Several other instance variables are defined in these flavors to enable the fault tree to be displayed graphically. The GATE flavor stores a special traversal of all events below and including the particular gate in a variable called :display-trav to be used by the graphical tree editor. NODEs, AND-GATEs, and OR-GATEs include a variable called :flavor-type that contains a description of the defining flavor that is displayed by the tree editor with other information about the events. The TREE flavor stores the name of each object in :name, which is also displayed. The TREE flavor has a component flavor, GWIN:BASIC-GRAPHICS-MIXIN, that furnishes all the information needed for the creation of the graphics objects that describe the fault tree. The tree editor will be described in the next sections along with the tree reduction algorithms. Figure 1 contains the actual fault tree flavor definitions written in LISP.

```
(DEFFLAVOR TREE
                ((parent nil)
                 (name nil)
                 (dependent nil))
                (gwin:basic-graphics-mixin)
        :settable-instance-variables)

(DEFFLAVOR GATE
                ((children nil)
                 (unavailability nil)
                 (dependent-eval nil)
                 (display-trav nil))
                (tree)
        :settable-instance-variables)

(DEFFLAVOR AND-GATE
                ((flavor-type 'AND-GATE))
                (gate)
        :settable-instance-variables)

(DEFFLAVOR OR-GATE
                ((flavor-type 'OR-GATE))
                (gate)
        :settable-instance-variables)

(DEFFLAVOR NODE
                ((flavor-type 'NODE)
                 (unavailability *default-unavailability*))
                (tree)
        :settable-instance-variables)
```

Figure 1. Fault tree flavor definitions.

Fault Tree Reduction Techniques

The evaluation of the fault tree is performed using a combination of standard fault tree reduction procedures. A bottom-up procedure is used for subtrees that do not contain repeated events, and a top-down, recursive procedure is used to evaluate subtrees that do contain repeated events. The tree is dynamically modularized according to the event which is being evaluated at the time. The locations of repeated events are propagated up the tree and stored in each event object. This information is used to determine which evaluation procedure is required for each event, and intermediate results are stored as they are calculated. These algorithms are described in detail in [5]. The object-oriented approach to fault tree reduction greatly increases the efficiency of the evaluation algorithms.

Graphical Fault Tree Editor

A general purpose graphical tree editor program that is available in Explorer System 3.0 was adapted to display the fault tree objects. The graphical display of the fault tree enables a visual check of the input tree structure. The user-interface, which includes menu-driven tree editing and pop-up displays of nodal data, allows the user to quickly and easily check the information stored in each fault tree object. Display of the tree after evaluation incorporates the results of tree reduction and modularization information. Probability values can also be edited with a series of menu-driven operations, and the modified tree can be re-evaluated.

The tree editor displays each fault tree object as a rectangle, with the name of the object in the center of the rectangle. GATE objects are shaded, and NODE objects are transparent. The interconnections of the tree objects are shown as lines connecting each node to its parent(s) and children. Figure 2 is a reproduction of a tree editor display as it appears on the Explorer screen. Features such as *zoom in*, *zoom out*, *move left*, and *move right* for tree positioning are accessed from a menu at the top of the display.

Object information other than the name of the object can be displayed by clicking on the specific object of interest. A pop-up window appears that contains the name of each instance variable associated with the object, and its value. Instance variables that are used by the tree editor only are not displayed. Figure 3 is a reproduction of the display after a GATE object is selected for the additional information. The pop-up window displays the GATE's flavor type, parent(s), children, unavailability, dependent list, and its dependent-eval list. The information displayed for a NODE includes flavor type, parent(s), and unavailability.

The probability value of any node in the fault tree can be changed in the graphical tree editor. By clicking on an object in a specified manner, a pop-up window appears that asks for a new probability value for that object. The value entered is then stored in the :unavailability variable for that object. In this way, errors in the object descriptions can be corrected before the tree is evaluated without having to exit the program and edit the data file. The tree may then be evaluated as usual. Modifications to tree objects are not reflected in the input file; however, they are documented in the output file each time the tree is evaluated.

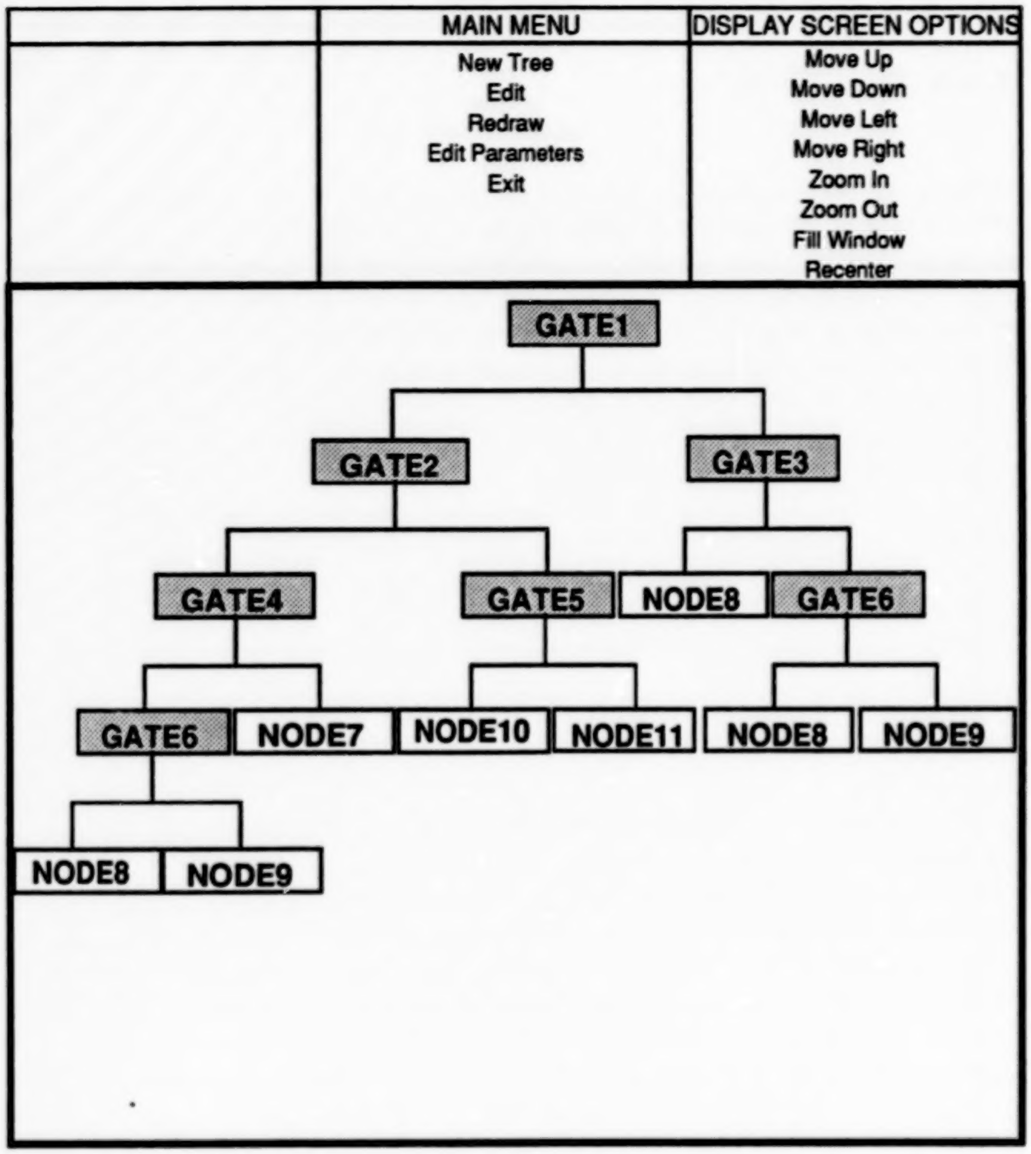
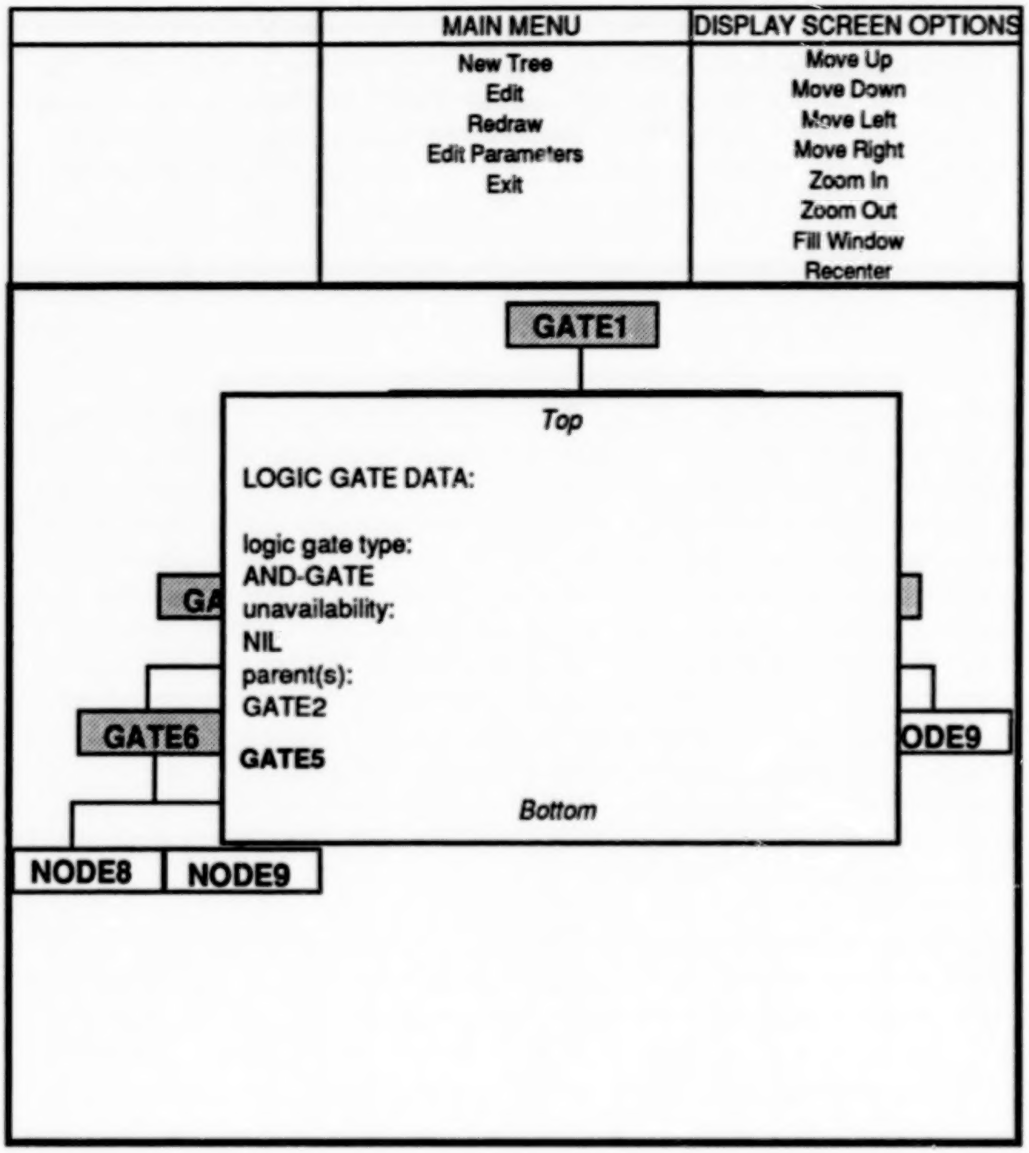


Figure 2 Display of a simple fault tree on the Explorer.



LOGIC GATE DATA:

logic gate type:
AND-GATE
unavailability:
NIL
parent(s):
GATE2

GATE5

Figure 3 Display of GATE data.

An interesting use of this editing capability is the variation of probability values in order to determine the effects on the probability of occurrence of the top event. For example, a tree can be evaluated and displayed after evaluation. One or more probability values for any tree object, GATE or NODE, can be changed and the tree is then re-evaluated. When such changes are made, the results of the previous calculation are retained for branches that are not affected by the changes. Therefore, calculations are not repeated unnecessarily, and the re-evaluation is completed in as few calculations as possible. None of the most widely used fault tree codes have such editing capabilities.

Conclusions

Object-oriented fault tree techniques provide an improved and flexible environment for reliability analysis. System components are represented by objects which can be organized into a persistent knowledge base of reliability information, improving data consistency. The inheritance heirarchy inherent in the object-oriented environment allows data to be entered either by class for groups of similar components or individually for specific components. Fault trees can be displayed graphically, allowing tree structure to be checked visually. Reliability data for NODEs can also be checked in the tree editor and updated immediately if desired. The tree reduction algorithms perform a direct evaluation of the fault tree and store a probability of occurrence for each event in the event's object. These algorithms are more efficient than previous algorithms implemented in conventional programming languages. The object-oriented environment also enables parameter variation studies to be performed on-line in conjunction with the tree editor.

The flexibility of this environment and the improvements already apparent in the fault tree application suggest that object-oriented fault trees may be appropriate for improving fault detection and diagnosis in complex systems. This topic is currently being explored to provide fault management in the large knowledge-based systems required by space applications.

Acknowledgements

This research was performed while FAPH was a graduate student at the University of Texas at Austin and was sponsored by the NASA Graduate Student Researchers Program, ZONTA International, and NSF grant DMC-8615432.

References:

1. Fussell, J. B., "A Formal Methodology for Fault Tree Construction," *Nuclear Science and Engineeering,* vol. 52, p. 421-432, 1973.
2. Vesely, W. E., Goldberg, F. F., Roberts, N. H., and Haasl, D. F., Fault Tree Handbook, NUREG-0492, 1981.
3. Stefik, M., and Bobrow, D. G., "Object-Oriented Programming: Themes and Variations," *The AI Magazine,* vol. VI, no. 4, p. 40-62, Winter 1985.
4. Explorer Lisp Reference, Texas Instruments, Incorporated, 1987.
5. Patterson-Hine, F. A., Object-Oriented Programming Applied to the Evaluation of Reliability Fault Trees, Ph.D. Dissertation, The University of Texas at Austin, May 1988.

N89 - 15610

Considerations in Development of Expert Systems for Real-Time Space Applications

S. Murugesan
NASA Ames Research Center
Mail Stop: 244-4
Moffett Field, CA 94035

ABSTRACT

Over the years, demand on space systems has increased tremendously and this trend will continue for the near future. Enhanced capabilities of space systems, however, can only be met with increased complexity and sophistication of onboard and ground systems. Artificial intelligence and expert system techniques have great potential in space applications.

Expert systems could facilitate autonomous decision making, improve in-orbit fault diagnosis and repair, enhance performance and reduce reliance on ground support. However, real-time expert systems, unlike conventional off-line consultative systems, have to satisfy certain special stringent requirements before they could be used for onboard space applications. Challenging and interesting new environments are faced while developing expert system space applications.

This paper discusses the special characteristics, requirements and typical life cycle issues for onboard expert systems. Further, it also describes considerations in design, development, and implementation which are particulary important to real-time expert systems for space applications.

1. INTRODUCTION

Current and future generation space systems are called upon to perform complex and more sophisticated, and intelligent tasks. The complexity of these systems are increasing in three dimensions: (1) the number of functions to be monitored and controlled, and also the kinds and volume of data to be considered, (2) need for quick response and faster rate of processing, and (3) need for more intelligent behavior.

There is a growing interest and pressing need for using knowledge-based problem solving techniques to cope with the increased demands on aerospace systems. Proper application of these techniques can provide better strategies for solving complex problems as discussed by Heer and Lum [14], and Weinweber [18]: i) autonomous satellite and space station control, ii) electric, propulsion, life support and thermal subsystems fault diagnosis, in-orbit repair/reconfiguration, and servicing, iii) intelligent vision and robotic systems with ability to recognize objects and scenes, and to find their way in places, whose conditions are not known, and far from earth.

Real-time expert systems are appropriate where there is an inherent need to enhance system autonomy without human assistance/intervention, where conventional techniques cannot make use of all relevant information providing intelligent or optimal solutions within a given time. Also, they could be used where humans work under severe psychological tensions, suffer from cognitive overload, fail to effectively monitor and evaluate all available information quickly, or make high-cost mistakes. Further, real-time expert systems are finding applications in domains such as routine operation and control, where qualified personnel who are able to evaluate complex situations and recommend actions are scare and not available all the time. A good discussion on the need for and desired features of expert system to assist human operators in monitoring and control of complex real-time process is given by Dvorak [8].

PRECEDING PAGE BLANK NOT FILMED

2. CHARACTERISTICS OF REAL-TIME SYSTEMS

The term real-time is often easier to recognize than to define. Though we understand the meaning of the term within the context of our own work, there is no consensus on a general or global definition. Also, there is a lot of misconception about what is meant by real-time. Some think that a system is considered real-time if it processes data quickly [26]. Another common usage is that real-time means "perpetually fast".

The feature that defines a real-time system is the system's ability to recognize an external event and to give a response by performing a service within a prescribed fixed time, which is dictated by application environment and criticality of the event. Response time - the time computer/expert system takes to recognize and respond to an external event -, is the most important factor in real-time applications. If events are not controlled within the allowed time, the process might go out of control and result in catastrophic effects. If given an arbitrary input or event, and an arbitrary state of the system, the system always produces a response by the time it is needed, then the system is said to be real-time [18]. The desired response time might vary from a few hundreds of microseconds to a few seconds, depending on application. Also, real-time expert systems have to perform their functions continually without ignoring on-going processes. Real-time systems are also known as *interrupt driven* and *reactive* systems. When an interrupt mechanism is used to signal a request for service/attention, the program/system becomes non-deterministic in that it is not possible to predict exactly what it will be doing a given number of clock cycles after initialization of the system.

In expert systems commonly used for medical diagnosis, design, configuration, financial analysis/advice, and other similar applications data is static and time to respond or give a decision is not critical. These off-line (sometimes called '*soft* real-time') advisory systems operate in non-dynamic domains at static points in time (i.e. data base, knowledge base, decision rules, etc. are fixed during a decision process). They do not have to keep up with the rapidly changing events in the external world, or meet high standards of '*hard*' real-time systems used for critical applications, such as very high reliability, availability and recovery after crashes.

The real-time domains have the following special characteristics, posing a set of complex and challenging problems for design and development of real-time expert systems [18].

- **Dynamic data (non-monotonocity):** Incoming sensor data, as well as facts that are deduced, do not remain static during the entire run of the program. They may decay in validity with time or they cease to be valid because external events have changed the state of the system.
- **Guaranteed response time:** The system must be able to respond by the time response is needed. Further more, one would like to achieve best response time within the deadline. Also, behavior of the expert system should be predictable that the response will fall within bounds or constraints.
- **Asynchronous inputs/interrupts:** Real-time systems must be capable of accepting asynchronous inputs and interrupts from external events. Also, they must be capable of interrupting an ongoing decision making process and resuming it after higher priority tasks are processed. It must have interfaces to gather data from a set of sensors or other (expert) systems. These requirements make testing and verification of expert systems more difficult.
- **Temporal reasoning:** Time is naturally an important variable in real-time domains. Typically, a real-time system needs the ability to reason about past, present and future (expected or anticipated) events, as well as the sequence in which the events had occurred. Therefore, knowledge representation schemes should permit representation of temporal relationships. A facility should exist for maintaining, accessing and statistically evaluating historical data.

- **Integration with procedural components:** Must be capable of integrating with conventional real-time software, which performs tasks such as data compression, signal or data processing, feature extraction and other application specific inputs/outputs.

- **Focus of attention mechanism:** When a significant event occurs, a real-time system should be able to reprioritize it goals and focus on important goals first. It could involve assigning context in which certain rules apply, modifying the set of sensors the system is currently looking at and changing the rate at which data is being analysed.

- **Continuous operation:** It must be capable of continuous operation over a long time - until stopped by an operator (through commands) or by other specific external events. Close attention, therefore, must be paid to garbage collection (efficient recycling of memory elements that are no longer needed) and archiving (maintenance) of sensor histories as far back as rules require them. Further, garbage collection must be done 'on the fly', not at processor's discretion.

- **Explanation facility:** It shows how the expert system reached a given conclusion and why the conclusion is justified. It gives a sequence of rules and facts that lead to a particular conclusion and describes its rationale for doing so. It is like an argument in favour of the conclusion. It is very essential in operator-assisted critical real-time systems, since the operator can accept or over-rule the decision reached by the system, after looking at the explanations by the system. Also, it can be used effectively for debugging and maintenance (extension) of knowledge base and to inspire confidence in systems performance and reasoning process [33].

In addition, constraints on realization of reliable and radiation-tolerant systems for space applications using microcomputers, and their processing speed and memory size limitations, is a critical bottleneck in applying knowledge-based techniques to real-time domains. Innovative methods have to be followed to overcome this bottleneck. The following characterstics are especially important to space applications.

- **Robustness:** It refers to gracefully degraded, reather than abrupt or fragile, behaviour of expert systems, while dealing with problems at the periphery of its domain (knowledge).

- **Handling uncertainty or missing data:** The system must be capable of handling reasonably and safely the uncertain, incomplete, vague, and missing information.

- **Reliability:** Extremely high reliability of operation in the targeted application environment and high degree of correctness and consistency of decisions are very crucial.

- **Fault tolerance:** Tolerance to failures of hardware, software (knowledge base, inference engine, operating system, etc.), and monitoring devices; fail-safe operations and graceful degradation

- **Ease of verification, validation and testing:** The system should be designed such that it is easily testable under various operational modes, and credible contingencies. Thorough verification and validation, and demonstration of proper functioning is very essential before actual use.

3. DESIGN AND DEVELOPMENT

Development of expert systems should be considered as a system engineering activity encompassing many tasks. It is a "team work". The division of expert system life cycle into various phases reduces the complexity of design by grouping and ordering main tasks of development [29]. It provides guidance on the order in which a project should carry out its major tasks. Many projects have come to grief, exceeded budget and schedule, and/or didn't deliver what was required, because they pursued their various development and evolution phases in a wrong way. Division of life cycle also helps to enforce an accepted development methodology among various persons involved in different phases and areas of development. Major phases in life cycles of expert systems development include:

- Problem identification/specification
- Acquisition of domain knowledge from experts, documents, previous case history, etc.
- Formulation of knowledge base, knowledge representation
- Choice (and/or development) of suitable inferencing/reasoning schemes and procedures
- Testing of expert system software (residing in development tools) under static (non-real-time) and real-time environments
  — Review human domain experts and specialists; revisions
- Integration of hardware deliverables and complied 'expert software'
- Testing under simulated and real-life environments under various modes of operation
  — Reviews by human domain experts and specialists; revisions
- Verification and validation: It covers the entire life cycle, and not just testing before delivery
- Delivery of flight-worthy Expert System; maintenance, upgrading and evolution

A typical life cycle of expert system is given in Figure 1. End product and outcome of each phase of development of expert system is summarized in Table 1.

3.1 Problem identification and domain feasibility study

Expert systems are useful for solving well-formulated problems, for which algorithmic solutions donot exists. These problems could be solved by using predetermined methods and heuristics that human experts have accumulated over years of learning and experience. An expert system consists of two basic elements [13]: a knowledge base, and an inference engine. A knowledge base consists of formalized facts and heuristic in a specific problem-solving domain. Inference engine uses this knowledge to solve problems.

The study of suitability of a domain for expert system application is very important and it involves the following steps [40]: 1) determine the nature of task, 2) determine if experts, who can solve the problem and are willing cooperative to share their expertise, exists and 3) determine whether their expertise can be modelled via an expert system. Domains suitable for expert system application tends to be deep and narrow. If the problems lend themselves to numeric or algorithmic solutions, it would be more effective to use those methods, rather than expert systems. However, if solution involves more of heuristics and human experts can solve the problem within a reasonable time and explain the solution process, expert systems could be considered. In essence one has to see whether the problem is "do-able" by one or more expert systems.

3.2 Requirements engineering

Expert system system development should begin with a complete, consistent and unambiguous idea of the needs of the user or the requirements of the system, and they should be well documented. The main advantages of the requirements analysis and understanding are: It serves as common ground for agreement between the developer and user/coustomer and helps in avoiding misunderstanding between them. It helps in early detection of errors. A survey [1] indicates that about thirty percent of errors in a major software intensive projects are due to faulty requirement specifications. It also helps the programmers to check that all the requirements are met. Further, it helps in defining requirements and specifications of the various real-time interfaces. In addition, it helps in generating good test cases and judging the quality of test cases actually used. Requirements of the system should be reviewed prior to the next phase of activity, by experts and project managers, system integrators, developers of other interfacing systems and must be agreed upon. The persons who are actually going to use the system during the operational phase should also be involved in this task.

3.3 Knowledge acquisition

The essence of an expert system is acquiring and encoding knowledge about a domain and then using it to solve problems in that domain. Knowledge elicitation has been cited as one of the bottlenecks in expert system development. Knowledge acquisition is the transfer of problem solving expertise from several sources, which include human experts, text books, literature, data bases, case histories and previous experiences. Of all these sources, the expertise of human specialists forms the main target of knowledge acquisition. Various techniques for acquiring knowledge from experts can be found in [7, 25, 27].

3.4 Knowledge representation

Knowledge representation refers to structuring of the acquired knowledge into computer recognizable form. Several knowledge representation models [13, 33] such as rules, semantic networks, and frames are being used, and each model has both advantages and disadvantages depending upon characteristics of the domain knowledge. Also, there are schemes which use a mixture of these representation schemes, to gain maximum benefits.

Various principles of software engineering [30, 31, 40] could be used in the design of expert system. They include: information hiding, separation of concerns, layering, and modularity. The principle of *information hiding* suggests that the group of rules hide internal details about the system. Also, while designing rules, it is better to separate different functionalities and use different rules to implement these functions. It is called the principle of *separation concerns.*

The principle of layering suggests that system should be considered as composition of layers. Any layer is aware of only the layer beneath it. The activation of rules at the higher level due to activation of rules at the lower level should be minimal. Such practice not only simplifies implementation, but also simplifies testing.

3.5 Testing and validation

"A system can best be designed if testing is interlaced with designing instead of being used after the design". The purposes of testing and evaluation include: i) guaranting satisfactory performance of the system, ii) locating weaknesses in the system, so that further improvement can be done by making knowledge-base richer and problem solving strategies more powerful, iii) evaluating different functions, and iv) evaluating correctness of the results, response time, etc. Testing creates tangible degree of trust in system reliability. However, difficulties in defining the correct test strategy causes errors to go undetected. Functionality and design bugs not caught during development testing have been found after prolonged time when the system is in actual use.

There are two basic approaches to testing: *Block box* and *white box* methods. The block box approach is based on specifications and functional requirements analysis and input output characteristics [16]. The white box method is driven by the way the system (rules, inference strategies, etc.) are designed and implemented. The block box testing is performed by generating a test case (a set of test input) [11]. Test case is generally prepared manually based on specification of the system and when available from real-life data. The white box testing focuses on correctness of implementation, without much regard to overall system functionality. A combination of both the black box and white box techniques would be most effective.

By documenting test comprehensiveness goals in test plans one can lower the probability of missing tests (see IEEE Standard 829). Test comprehensiveness are defined in terms of four types of coverage: requirements, input domain, output range, and structure. Most probable errors in testing process are: 1) creating too few tests (they leave many bugs undetected in the delivered system), 2) creating wrong tests (they detect 'wrong bugs' rather than 'right bugs' that are critical and cause serious trouble), and 3) creating too many tests (doing unnecessary and redundant checks. Redundant testing can be avoided if one finds the test coverage each test provides). The IEEE standard 1008 for software unit testing gives some guidelines for testing.

Verification and validation (V&V) of expert system is one of the very important and difficult task of development of expert systems. Verification refer to confirming that the system has been developed correctly according to accepted methodology and system requirements, while validation means ensuring that system correctly serves the purpose for which it is intended. A V&V process is expected to catch user input errors, incorrect rules and facts, redundant rules, incorrect behavior of inference engine, and incorrect output after having reached a correct conclusion. 'V&V is not just a one day concern just before testing or delivery of a system; it spans the entire life cycle of expert system'. Currently , there is a lot of interest and concern in this area, and more information can be found in [3, 5,6 9, 10, 21-24, 28 36, 37].

4. MAINTENANCE OR SYSTEM EVOLUTION

Many expert system applications are characterized by lack of consistent and complete knowledge at the representation level, especially at the beginning of a project. Hence, it becomes necessary to modify existing knowledge base continually and maintain its consistency as new knowledge is imparted. Though this is a very important part of expert system development and operation [34], it is often gets least attention. Unlike in conventional software, rules and knowledge about the domain evolve with experience of their use, and hence, may have to be modified more often than algorithms. Belief support of rules might vary with feedback from earlier decisions. Further, validity of some rules, which might be time-dependent, can change over time, necessitating modification. Also, modification becomes necessary to correct errors found during various phases of development and during actual operation in the targeted environment.

Thus, as given by Ramamurthy [29], expert system maintenance could be *perfective* which encompasses changes asked by the user, *adaptive* which encompasses changes in environment, and *corrective* which corrects undiscovered errors and mistakes. In larger systems, about 65 percent of maintenance is perfective, 18 percent is adaptive, and 17 percent is corrective [20].

Mostly maintenance of expert system have to done by people not involved in the original development and hence they have to learn first about the system they are planing to maintain. This calls for better clarity, accuracy and completeness of different kinds of documents, besides skill and experience of people concerned. The more difficulty to understand the system, the more difficult it is to maintain, and hence, higher the maintainability risk. Many people prefer to call this phase as "system evolution" or "system enhancement" phase, rather than "maintenance".

Most expert systems, being large software projects, will suffer from what is known as *deadline effect*, limiting maintainability and reusability [29]. Most projects have completion deadlines. In the debugging and testing phase, with deadlines near, top priority of developers is to fix errors/bugs. The worst part is that many difficult bugs tend to get detected near deadlines. This forces to resort to "quick and dirty" fixing and hence systems loose their maintainability. Such practices have to be avoided.

5. SUMMARY

Though expert systems have found wide spread use in many applications, their use in critical real-time applications are very few. Development of real-time expert systems are much more difficult than the traditional consultative and advisory expert systems. Further, testing and validation of them still remains as a major problem. Expert systems cannot solve all types of problem. It is very important to understand the scope and limitations of current expert systems technology (both hardware and software) for critical aerospace applications, which pose many constraints.

Knowledge elicitation is a very important activity and a lot of attention has to be paid to it, as the knowledge is the key to success of an expert system. Also, experts must be cooperative, invest time, and must help in testing the system. Good design, documentation, adherence to accepted development methodology, enforcement of discipline in program design and modifications, and thorough testing and validation are very important for successful operation of expert systems in space.

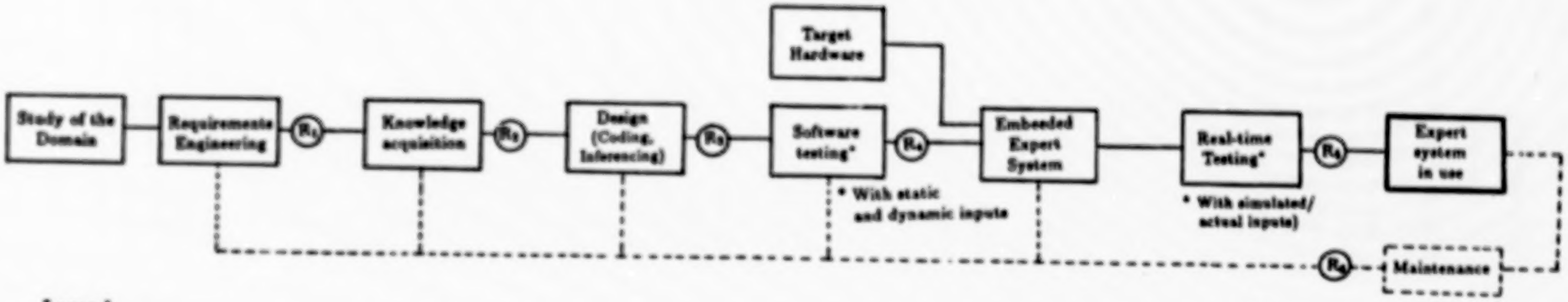
Acknowledgements

This work was done while the author held a National Research Council-NASA Ames Research Center Research fellowship. The author thanks Hamid Berenji and Terry Grant for their careful reviews and comments on draft of this article. He is also grateful to Dr. Henry Lum for his guidance and support.

References

1. Basili V.R, and Perricone, "Software errors and complexity: An emprical investigation", Comm. ACM, Jan. 1984
2. Berns G.M, "Assessing software maintainability", Communications of ACM, January 1984, pp 14-23
3. Boehm B.W, "Verifying and validating software requirements and design specifications, IEEE Software, Jan. 1984, pp 75-88
4. B.W. Boehm, "Spiral model of software development and enhancement," Computer, May 1988
5. Culbert C, et. al., "Approaches to verification of rule-based expert systems," SOAR'87, NASA-CP-2491, August 1987
6. Culbert C, et. al., "An expert system methodology which supports verification and validation," IEEE Conf. on AI Applications, 1987
7. Evanson S.E, "How to talk to to an expert," AI Expert, Feb. 1988, pp 36-41
8. Dvorak, "Expert systems for monitoring and control," Proc. Artificial Intelligence and Advance Technology Conf., Long Beach, Calif. April 1987.
9. Gaschnig J, et al., "Evaluation of expert systems: Issues and case studies," in D.A. Waterman, et al. (ed.) Building Expert Systems, Addison Wesley, 1983
10. Geissman J, "Verification and validation of Expert systems," AI Expert, Feb. 1988
11. Goodenough J.B, and Gerhart, "Toward a theory of test data selection," IEEE Trans. Software Engineering, Vol. SE-1, June 1975,
12. Green P. E., "Resource limitation issues in real-time intelligent systems," Proc. SPIE Conf. on Applications of artificial Intelligence, Vol. 635, Orlando, Fl, April 1986

13. Harmon P and King D, "Expert systems," John Wiley and Sons, New York, 1985
14. Heer L and Lum H, "Raising the AIQ of the space station," Aerospace America, Vol. , Jan. 1987, pp 16-17
15. Heny M.S.H, "Why evolutionary development," Future Generation Computing Systems, No.3, pp 103-109, 1987
16. Howden W.E, "The theory and practice of functional testing," IEEE Software Sep. 1985, pp 6-17
17. Irland E.A, "Assuring quality and reliability of complex electronic systems: Hardware and Software", Proc. IEEE, Vol. 76, No.1, Jan. 1988, pp 5-18
18. Laffey T.J, "Real-time knowledge-based systems", AI Magazine, Spring 1988, pp 27-45
19. Leinweber, D "Expert systems in space," IEEE Expert, Vol. 2, No.1, pp 26-36, 1987
20. Lientz B.P and Swanson E.B, "Software maintenance management", Reading, MA: Addision-Wesley, 1980
21. Miller E and Howden W.E, "Software testing and validation techniques," IEEE Computer Society Tutorial, 1981
22. Nguyen T, "Knowledge base verification," AI Magazine, Summer 1987
23. Nguyen T.A, "Verifying of consistency of production systems," Proc. Third conf. on AI Applications, Feb.1987, pp4-8
24. O'Keefe, et. al., "Validating expert system performance," IEEE EXPERT, Winter 1987
25. Olson J.R and Ruter H.H, "Extracting expertise from experts: Methods of knowledge acquisition," Expert systems, Aug. 1987, pp 152-168
26. O'Reiley, C.A, and Cromarty A.S, "Fast is not real-time: Designing effective real-time AI systems," Proc. SPIE 548, 1985, pp 249-257
27. Prerau D.S, "Knowledge acquisition in the development of large expert systems," AI Magazine, Vol. 8, Summer 1987, pp 43-51
28. Ramamurthy C.V, et. al, "Application of a methodology for development and validation of reliable process control software," IEEE Trans. Software Engineering, Vol. SE-7, No.7, Nov. 1981, pp 537-555
29. Ramamurthy C.V, et.al.," Programing in large," IEEE trans. Software Engineering, Vol. SE-12, No.7, July 1986, pp 769-783
30. Ramamurthy C.V, et. al., "Software development support for AI programs," Computer, Jan. 1987, pp 30 - 40
31. Reeker L.H, et.al., "Applying software engineering to knowledge engineering (and vice-versa), 27th Annual Technical ACM Symposium, Washington D.C., 1988
32. Shirely R.S, "Some lessons learned using expert systems for process control," IEEE Control systems Magazine, Dec. 1987, pp 11-15
33. Special issue on knowledge representation, Computer 1983
34. Soloway, E, et.al. "Assessing the maintainability of XCON: Coping with the problems of a very large rule-base," Proc. AAAI-1987, Seattle, WA, July 1987
35. Sorrells, "Time-constrained inference strategy for real-time expert systems," IEEE Proc. WESTEX 1985, pp 1336-1341
36. Stachowitz R.A, et. al. "Building validation tools for knowledge-based systems," First annual workshop on Space Operations Automation and Robotics, SOAR'87, NASA CP-2491, 1987
37. Suwa M, et al. "An approach to verifying completeness and consistency in a rule based expert system," AI Magazine, Fall 1982, pp16-21
38. Turner M, "Real-time experts", System International Vol. 14, No.1, 1986 pp 55-57
39. Wright, M, et.al., "An expert system for real-time control," IEEE Software, March 1986, pp 16-24
40. Zualkernan, et.al., "Expert systems and software engineering: Ready for marriage?," IEEE Expert, Winter 1986, pp 25-31



Legend:
R_1: Requirements review
R_2: Review of domain knowledge/expertise
R_3: Design review
R_4: Software review, evaluation, validation
R_5: Test review, validation
R_6: Review of request for modification/enhancement

Fig.1 Life cycle of an expert system

Table I: Outcome of each phase of life cycle of an expert system

Phase of life cycle	**Outcome**
Study of suitability of the domain	Whether the problem can be tackled
Requirement Analysis/Engineering	Clear understanding of needs and requirements of targeted system, Interface specifications Documents: – *Functional Requirements Specifications* – *Test requirements - A preliminary study*
Requirements Review	
Knowledge Acquisition	Documented expertise (in natural language)
Knowledge verification & Review	Certified expertise
Coding Knowledge & Desiging problem solving strategies (Inference Engine)	Knowledge-based system design *Design document*
Design Review	
Testing in development environment under off-line simulated conditions	Removal of bugs/errors, inconsistencies *Test report, record of modifications*
Testing under real-time simulated environments	Checks and verifies time-dependent features - synchronization, response time, etc. *Test report, record of modifications*
Validation & Review	Certified Compiled knowledge
Transport to target hardware	Embedded expert system
Testing embedded system under simulated (static and dynamic) environments	
Testing under actual environment & Validation	Flight-worthy Expert System
Test and readiness Review	*Final documents with relevant revisions*
Flight operation (Maintainance and reuse)	Desired operations - Feedback for correction, further improvement - Review & authorisation of modifications Document update **Evolving expert system in use**

INDEX OF AUTHORS

Abernethy, Ken 451
Adamson, J. M. 163
Anderson, Audie 125
Armstrong, William W. 289
Asdjodi, Maryam 131

Barry, John M. 193
Bell, Benjamin 349
Bharwani, S. 93
Biegl, C. 243
Blanchard, Mary 253
Blevins, E. 93
Blokland, W. 243
Bond, W. E. 29, 271
Bosworth Jr., Edward L. 57
Brady, Mike 125
Bridges, Susan 323
Busse, Carl 356

Casadaban, Cyprian 183
Cellier, Francois 313
Chang, Chin-Liang 191
Chen, Chu Xin 233
Collard, Philippe E. 411
Colombano, Silvano 371
Combs, Jacqueline 191
Cook, G. E. 303

Das, A. 69
Davis, Stephen 205
Deugo, D. 431
Dutta, Soumitra 95

Eilbert, James L. 349

Fennelly, A. J. 57
Fernandez, K. 303
Fiala, Harvey E. 381
Foo, Norman Y. 261
Ford, Donnie 125
Freeman, Michael S. 39, 59
Friend, Robyn C. 391

Gerstenfeld, Arthur 75
Gettig, Gary A. 47
Goforth, Andre 411
Gorney, D. J. 457
Graves, Sara J. 173
Groundwater, B. 67

Hacke, Keith 271
Hays, Dan 107
Heinsheimer, Thomas F. 391
Hooper, James W. 131
Hung, Chaw-Kwei 19
Hydrick, Cecile L. 173

Johannes, James D. 141, 323, 331
Jones, J. U. 97

Kao, Simon M. 369
Karsai, G. 115, 303
Kiss, P. A. 59
Koen, B. V. 477
Koons, H. C. 457
Krishnamurthy, C. 243

Laffey, Thomas J. 369
Lee, Chung-Mong 231
Lee, Sung Yong 107
Lembeck, M. F. 67
Loda', Antonio G. 221
Loganantharaj, Raj L. 151

MacDonald, James R. 141
Marapane, Suresh 233
Marshall, G. 163
Matlin, Sam 421
McKee, James W. 85
Mohamed, Ahmed S. 289
Morris, Keith 245
Murugesan, S. 1, 487
Mutammara, A. 303

Nogueira, C. A. M. 441

Oliveira, C. A. 441
Oliveira, P. P. B. 441
Oppacher, F. 431
O'Reilly, Daniel 467

Padalkar, S. 115
Patrick, Clint 125
Patterson-Hine, F. A. 477
Pong, Ting-Chuen 231
Prince, Mary Ellen 331
Pulaski, Kirt 183

Ranganath, Heggere S. 221
Rao, Anand S. 261
Read, Jackson Y. 369
Rockowiak, Daniel 341
Rodrigues, V. 441
Rosenthal, Don 371
Rozenblit, Jerzy W. 313
Ruokangus, Corinne C. 5

Sabharwal, C. L. 271
St. Clair, D. C. 29, 271
Sarsfield, L. 67
Sary, Charisse 193
Schmidt, James L. 369
Schroer, Bernard J. 153
Shiva, S. G. 97
Siegel, Neil G. 391
Simoni, P. O. 441
Slagle, James 231
Springfield, J. 303
Stachowitz, Rolf 191
Sterling, Leon 211
Stock, Todd 191
Sticklen, Jon 29
Sztipanovits, J. 115, 243, 303

Tillotson, Brian 281
Trivedi, Mohan M. 233
Tseng, Fan T. 153, 207
Tyagi, Rajesh 207

Vezina, James M. 211

Walls, J. 93
Wang, Caroline 125
Weitzenkamp, Scott M. 369
Williams, Robert 467
Wogrin, Nancy 371
Wolfsberger, John W. 85, 107, 153
Wright, R. Glenn 253
Wu, Chuan-lin 19

Yarbrough, Kevin 467
Yeh, Show-Way 19
Young, Laurence 371

Zander, Carol S. 401
Zeanah, Hugh 125
Zeigler, Bernard P. 313
Zhang, S. X. 153

1. REPORT NO.
NASA CP-3013

2. GOVERNMENT ACCESSION NO.

3. RECIPIENT'S CATALOG NO.

4. TITLE AND SUBTITLE
Fourth Conference on Artificial Intelligence for Space Applications

5. REPORT DATE
October 1988

6. PERFORMING ORGANIZATION CODE

7. AUTHOR(S) Compiled By S. L. O'Dell, J. S. Denton, and M. Vereen

8. PERFORMING ORGANIZATION REPORT #

9. PERFORMING ORGANIZATION NAME AND ADDRESS
George C. Marshall Space Flight Center
Marshall Space Flight Center, Alabama 35812

10. WORK UNIT NO.
M-599

11. CONTRACT OR GRANT NO.

13. TYPE OF REPORT & PERIOD COVERED
Conference Publication

12. SPONSORING AGENCY NAME AND ADDRESS
National Aeronautics and Space Administration
Washington, DC 20546

14. SPONSORING AGENCY CODE

15. SUPPLEMENTARY NOTES Conference Director: J. S. Denton, Information and Electronic Systems Lab., Marshall Space Flight Center.
Co-sponsored by the University of Alabama in Huntsville.

16. ABSTRACT

Proceedings of a conference held in Huntsville, Alabama, on November 15-16, 1988. The Fourth Conference on Artificial Intelligence for Space Applications brings together diverse technical and scientific work in order to help those who employ AI methods in space applications to identify common goals and to address issues of general interest in the AI community. Topics include the following: space applications of expert systems in fault diagnostics, in telemetry monitoring and data collection, in design and systems integration; and in planning and scheduling; knowledge representation, capture, verification, and management; robotics and vision; adaptive learning; and automatic programming.

17. KEY WORDS
Artificial Intelligence
Computer Vision
Design Data Capture
Robotics
Space Station Automation

18. DISTRIBUTION STATEMENT
Unclassified/Unlimited

Subject Category: 61

19. SECURITY CLASSIF. (of this report)	20. SECURITY CLASSIF. (of this page)	21. NO. OF PAGES	22. PRICE
Unclassified	Unclassified	512	A22

For sale by National Technical Information Service, Springfield, Virginia 22161

NASA-Langley, 1988

END

DATE

FILMED

APR 10 1989